Deniable Key Exchanges for Secure Messaging

by

Nik Unger

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2015

This thesis consists of material all of which I authored or co-authored: see Statement of Contributions included in the thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Statement of Contributions

The content in Chapter 2 of this thesis was co-authored with Sergej Dechand, Joseph Bonneau, Sascha Fahl, Henning Perl, Ian Goldberg, and Matthew Smith. In particular, the expert usability reviews were performed by Sergej Dechand, Sascha Fahl, Henning Perl, and Matthew Smith, and the figures in Chapter 2 were produced by Sergej Dechand. All other chapters in this thesis contain original work authored under the supervision of Ian Goldberg.

Abstract

Despite our increasing reliance on digital communication, much of our online discourse lacks any security or privacy protections. Almost no email messages sent today provide end-to-end security, despite privacy-enhancing technologies being available for decades. Recent revelations by Edward Snowden of government surveillance have highlighted this disconnect between the importance of our digital communications and the lack of available secure messaging tools. In response to increased public awareness and demand, the market has recently been flooded with new applications claiming to provide security and privacy guarantees. Unfortunately, the urgency with which these tools are being developed and marketed has led to inferior or insecure products, grandiose claims of unobtainable features, and widespread confusion about which schemes can be trusted.

Meanwhile, there remains disagreement in the academic community over the definitions and desirability of secure messaging features. This incoherent vision is due in part to the lack of a broad perspective of the literature. One of the most contested properties is deniability—the plausible assertion that a user did not send a message or participate in a conversation. There are several subtly different definitions of deniability in the literature, and no available secure messaging scheme meets all definitions simultaneously. Deniable authenticated key exchanges (DAKEs), the primary cryptographic tool responsible for deniability in a secure messaging scheme, are also often unsuitable for use in emerging applications such as smartphone communications due to unreasonable resource or network requirements.

In this thesis, we provide a guide for a practitioner seeking to implement deniable secure messaging systems. We examine dozens of existing secure messaging protocols, both proposed and implemented, and find that they achieve mixed results in terms of security. This systematization of knowledge serves as a resource for understanding the current state-of-the-art approaches. We survey formalizations of deniability in the secure messaging context, as well as the properties of existing DAKEs. We construct several new practical DAKEs with the intention of providing deniability in modern secure messaging environments. Notably, we introduce Spawn, the first non-interactive DAKE that offers forward secrecy and achieves deniability against both offline and online judges; Spawn can be used to improve the deniability properties of the popular TextSecure secure messaging application. We prove the security of our new constructions in the generalized universal composability (GUC) framework. To demonstrate the practicality of our protocols, we develop and compare open-source instantiations that remain secure without random oracles.

Acknowledgments

Foremostly, I would like to thank Ian Goldberg, as this thesis would not be possible without his outstanding supervision. I would like to thank Urs Hengartner and Doug Stinson for their excellent feedback. I would also like to thank my other co-authors: Sergej Dechand, Joseph Bonneau, Sascha Fahl, Henning Perl, and Matthew Smith; together we produced source material for Chapter 2 of which we can all be proud. We thank the anonymous reviewers, Trevor Perrin, and Henry Corrigan-Gibbs for their helpful feedback on Chapter 2. Finally, I would like to thank the other members of the CrySP lab (and especially the inhabitants of `#crysp`) for fostering a fantastic creative environment.

The artwork on the quotation page was produced on commission by Danny Rivera; its use is governed by the Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License. The drawing has been modified from the original version.

Dedication

To my parents, for their motivation, love, and support.

Table of Contents

List of Figures

List of Tables

List of Algorithms

List of Definitions, Theorems, and Conjectures

There are gems of wondrous brightness
Ofttimes lying at our feet,
And we pass them, walking thoughtless,
Down the busy, crowded street.
If we knew, our pace would slacken,
We would step more oft with care,
Lest our careless feet be treading
To the earth some jewel rare.

RUDYARD KIPLING

Chapter 1

Introduction

For the past few decades, our society has become increasingly reliant on digital communication. Today, we make use of large-scale platforms such as the Internet, mobile networks, and leased communication lines to reliably deliver our most critical discourse. Despite the immense importance of these communication channels, a significant portion of our transmitted messages are completely free of security or privacy protections. For example, nearly all email communications lack end-to-end security even though protocols such as OpenPGP and S/MIME have been available for decades; these schemes have failed to achieve widespread adoption and have been plagued by usability issues [WT99; GM05; GMS+05; RVR14]. Consequently, there is a prominent disconnect between the importance of our digital communications and the effort that users employ to secure them.

The lack of widely adopted secure communication tools can be attributed to several problems: a lack of computational resources for cryptography during the Internet's formative years, serious usability issues with available tools, and a general lack of concern about privacy issues. However, recent revelations about mass surveillance by intelligence services have highlighted the lack of security and privacy in our messaging tools and spurred demand for better solutions [Ope14a]. A recent Pew Research poll found that 80% of Americans are now concerned about government monitoring of their electronic communications. A combined 68% of respondents reported feeling "not very secure" or "not at all secure" when using online chat and 57% felt similarly insecure using email [Mad14]. With widespread availability of computational power and renewed interest in secure communication, new applications are being developed to meet this demand.

Unfortunately, many new messaging tools are failing to achieve their claimed security objectives. Despite the publication of a large number of secure messaging protocols in the

academic literature, tools are being released with new designs that fail to draw upon this knowledge, repeat known design mistakes, or use cryptography in insecure ways. However, the academic research community is also failing to learn some lessons from tools in the wild.

Furthermore, there is a lack of coherent vision for the future of secure messaging. Most solutions focus on specific issues and have different goals and threat models. These problems are compounded by differing security vocabularies and the absence of a unified evaluation of prior work. Outside of academia, many products mislead users by advertising with grandiose claims of "military grade encryption" or by promising impossible features such as self-destructing messages [Gol14; Tel14; Wic14; Con14]. The recent EFF Secure Messaging Scorecard evaluated tools for basic indicators of security and project health [Ele14] and found many purportedly "secure" tools do not even attempt end-to-end encryption.

A widespread weakness in current secure messaging tools is the lack of strong deniability properties. Deniable secure messaging schemes allow conversation participants to later plausibly deny sending messages, or even participating in a conversation. This notion was popularized in the secure messaging context with the release of Off-the-Record Messaging (OTR) a decade ago [BGB04]. Unfortunately, the OTR protocol is not well suited to modern settings such as mobile device communication due to its requirement for synchronous connections. Protocol designers seeking to achieve OTR-like deniability properties in these environments have been forced to turn to the cryptographic literature, and have found that existing primitives are not well suited to the task. Some practitioners have also prematurely dismissed deniability as an impractical property for modern secure messaging applications.

Our goal in this work is to facilitate new secure messaging research by providing a broad perspective of the field, and new tools for protocol designers. We aim to identify where problems lie and create a guide to help move forward on this important topic. We also approach the problem of deniability from a practitioner's perspective, and construct new cryptographic protocols designed to address modern secure messaging problems.

The primary contributions of this thesis are:

1. a systematization of knowledge of secure messaging schemes:
    - establishment of a set of common security and privacy feature definitions for secure messaging;
    - systematization of secure messaging approaches based both on academic work and "in-the-wild" projects;

- a comparative evaluation of these approaches;
- identification and discussion of current research challenges, indicating future research directions;

2. construction of several new cryptographic protocols for use in deniable secure messaging tools:
   - a dual-receiver encryption scheme that improves the practical performance of an existing deniable key exchange protocol;
   - a highly efficient deniable key exchange protocol designed for use in interactive settings (e.g., instant messaging);
   - an interactive deniable key exchange protocol requiring only a single communication round;
   - a non-interactive key exchange protocol with forward secrecy and the strongest deniability properties ever achieved in this setting;
   - a method to incorporate our non-interactive key exchange into a popular secure messaging application for smartphones;
   - security proofs for our newly constructed protocols;
3. development of open-source implementations of our new cryptographic protocols.

Chapter 2 presents our systematization of knowledge of current secure messaging approaches. Chapter 3 describes the problem of deniability in the secure messaging context, and includes our new cryptographic constructions. Chapter 4 provides an overview of our implementations and performance evaluations. Finally, Chapter 5 includes some closing remarks.

Chapter 2

Secure Messaging

This chapter is adapted from work that previously appeared in the 2015 IEEE Symposium on Security and Privacy [UDB+15a], and later as a technical report published by the Centre for Applied Cryptographic Research [UDB+15b].

Motivated by recent revelations of widespread state surveillance of personal communication, many products now claim to offer secure and private messaging. This includes both a large number of new projects and many widely adopted tools that have added security features. The intense pressure in the past two years to deliver solutions quickly has resulted in varying threat models, incomplete objectives, dubious security claims, and a lack of broad perspective on the existing cryptographic literature on secure communication.

In this chapter, we evaluate and systematize current secure messaging solutions and propose an evaluation framework for their security, usability, and ease-of-adoption properties. We consider solutions from academia, but also identify innovative and promising approaches used "in the wild" that are not considered by the academic literature. We identify three key challenges and map the design landscape for each: *trust establishment*, *conversation security*, and *transport privacy*. We aim to establish evaluation criteria for measuring security features of messaging systems, as well as their usability and adoption implications. A further goal in this chapter is to provide a broad perspective on secure messaging and its challenges, as well as a comparative evaluation of existing approaches, in order to provide context that informs future efforts.

After defining terminology in Section 2.1, we present our systematization methodology in Section 2.2. In subsequent sections (Section 2.3, Section 2.4, and Section 2.5), we evaluate each of the proposed problem areas. Our findings are discussed and concluded in Section 2.6.

2.1 Background

Secure messaging systems vary widely in their goals and corresponding design decisions. Additionally, their target audiences often influence how they are defined. In this section, we define terminology to differentiate these designs and provide a foundation for our discussion of secure messaging.

2.1.1 Types of specification

Secure messaging systems can be specified at three different broad levels of abstraction:

Chat protocols: At the most abstract level, chat protocols can be defined as sequences of values exchanged between participants. This mode of specification deals with high-level data flows and often omits details as significant as the choice of cryptographic protocols (e.g., key exchanges) to use. Academic publications typically specify protocols this way.

Wire protocols: Complete wire protocols aim to specify a binary-level representation of message formats. A wire protocol should be complete enough that multiple parties can implement it separately and interoperate successfully. Often these are specific enough that they have versions to ensure compatibility as changes are made. Implicitly, a wire protocol implements some higher-level chat protocol, though extracting it may be non-trivial.

Tools: Tools are concrete software implementations that can be used for secure messaging. Implicitly, a tool contains a wire protocol, though it may be difficult and error-prone to derive it, even from an open-source tool.

2.1.2 Synchronicity

A chat protocol can be *synchronous* or *asynchronous*. Synchronous protocols require all participants to be online and connected at the same time in order for messages to be transmitted. Systems with a peer-to-peer architecture, where the sender directly connects to the recipient for message transmission, are examples of synchronous protocols. Asynchronous protocols, such as SMS (text messaging) or email, do not require participants to be online when messages are sent, utilizing a third party to cache messages for later delivery.

Due to social and technical constraints, such as switched-off devices, limited reception, and limited battery life, synchronous protocols are not feasible for many users. Mobile environments are also particularly prone to various transmission errors and network interruptions that preclude the use of synchronous protocols. Most popular instant messaging (IM) solutions today provide asynchronicity in these environments by using a *store-and-forward* model: a central server is used to buffer messages when the recipient is offline. Secure messaging protocols designed for these environments need to consider, and possibly extend, this store-and-forward model.

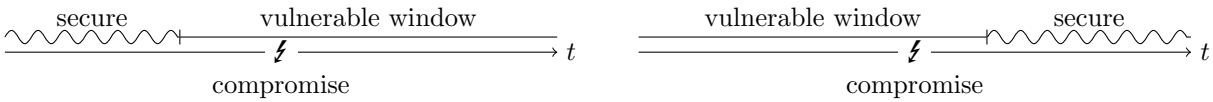
2.1.3 Deniability

Deniability, also called *repudiability*, is a common goal for secure messaging systems. Consider a scenario where Bob accuses Alice of sending a specific message. Justin, a judge, must decide whether or not he believes that Alice actually did so. If Bob can provide evidence that Alice sent that message, such as a valid cryptographic signature of the message under Alice's long-term key, then we say that the action is *non-repudiable*. Otherwise, the action is *repudiable* or *deniable*. We can distinguish between *message repudiation*, in which Alice denies sending a specific message, and *participation repudiation* in which Alice denies communicating with Bob at all. The high-level goal of repudiable messaging systems is to achieve deniability similar to real-world conversations.

The cryptographic literature has produced many subtly varying definitions of "deniability" since deniable encryption was first formally proposed [CDNO97]. For the purposes of this chapter, we consider message and participation repudiation only in the context of a judge that examines a protocol transcript after a conversation has concluded, attempting to determine if the transcript is genuine. We return to this issue in Chapter 3, where we consider stronger notions of deniability granting judges real-time access to malicious conversation participants.

2.1.4 Forward/Backward Secrecy

In systems that use the same static keys for all messages, a key compromise allows an attacker to decrypt the entire message exchange. A protocol provides *forward secrecy* if the compromise of a long-term key does not allow ciphertexts encrypted with previous session keys to be decrypted (Figure 2.1a). If the compromise of a long-term key does not allow subsequent ciphertexts to be decrypted by passive attackers, then the protocol is said to have *backward secrecy* (Figure 2.1b). However, tools with *backward secrecy* are still



(a) Forward Secrecy (b) Backward Secrecy

Figure 2.1: Forward vs. backward secrecy. Session keys are protected from long-term key compromise.

vulnerable to active attackers that have compromised long-term keys. In this context, the "self-healing" aspect of *backward secrecy* has also been called *future secrecy*. The terms are controversial and vague in the literature [And97; Shi00; Ope13a].

2.2 Systematization Methodology

Over the years, hundreds of secure messaging systems have been proposed and developed in both academia and industry. An exhaustive analysis of all solutions is both infeasible and undesirable. Instead, we extract recurring secure messaging techniques from the literature and publicly available messaging tools, focusing on systematization and evaluation of the underlying concepts and the desirable secure messaging properties. In this section, we explain our precise methodology.

2.2.1 Problem Areas

While most secure messaging solutions try to deal with all possible security aspects, in our systematization, we divide secure messaging into three nearly orthogonal problem areas addressed in dedicated sections: the *trust establishment* problem (Section 2.3), ensuring the distribution of cryptographic long-term keys and proof of association with the owning entity; the *conversation security* problem (Section 2.4), ensuring the protection of exchanged messages during conversations; and the *transport privacy* problem (Section 2.5), hiding the communication metadata.

While any concrete tool must decide on an approach for each problem area, abstractly defined protocols may only address some of them. Additionally, the distinction between these three problem areas is sometimes blurred since techniques used by secure messaging systems may be part of their approach for multiple problem areas.

2.2.2 Threat Model

When evaluating the security and privacy properties in secure messaging, we must consider a variety of adversaries. Our threat model includes the following attackers:

- **Local Adversary (active/passive)**: An attacker controlling local networks (e.g., owners of open wireless access points).
- **Global Adversary (active/passive)**: An attacker controlling large segments of the Internet, such as powerful nation states or large Internet service providers.
- **Service providers**: For messaging systems that require centralized infrastructure (e.g., public-key directories), the service operators should be considered as potential adversaries.

Note that our adversary classes are not necessarily exclusive. In some cases, adversaries of different types might collude. We also assume that all adversaries are participants in the messaging system, allowing them to start conversations, send messages, or perform other normal participant actions. We assume that the endpoints in a secure messaging system are secure (i.e., malware and hardware attacks are out of scope).

2.2.3 Systematization Structure

Section 2.3, Section 2.4, and Section 2.5 evaluate *trust establishment*, *conversation security*, and *transport privacy* approaches, respectively. For each problem area, we identify desirable properties divided into three main groups: *security and privacy features*, *usability features*, and *adoption considerations*. Each section starts by defining these properties, followed by the extraction of generic approaches used to address the problem area from existing secure messaging systems. Each section then defines and evaluates these approaches, as well as several possible variations, in terms of the already-defined properties. Concrete examples of protocols or tools making use of each approach are given whenever possible. The sections then conclude by discussing the implications of these evaluations.

In each section, we include a table (Table 2.1, Table 2.2, and Table 2.3) visualizing our evaluation of approaches within that problem area. Columns in the tables represent the identified properties, while rows represent the approaches. Groups of rows begin with a generic concept, specified as a combination of cryptographic protocols, followed by extension rows that add or modify components of the base concept. Whenever possible, rows

include the name of a representative protocol or tool that uses the combination of concepts. Representatives may not achieve all of the features that are possible using the approach; they are merely included to indicate where approaches are used in practice. Each row is rated as providing or not providing the desired properties. In some cases, a row might only partially provide a property, which is explained in the associated description.

For each problem area, we identify desirable properties in three main categories:

1. **Security and Privacy Properties**: Most secure messaging systems are designed using standard cryptographic primitives such as hash functions, symmetric encryption ciphers, and digital signature schemes. When evaluating the security and privacy features of a scheme, we assume cryptographic primitives are securely chosen and correctly implemented. We do not attempt to audit for software exploits which may compromise users' security. However, if systems allow end users to misuse these cryptographic primitives, the scheme is penalized.

2. **Usability Properties**: Usability is crucial for the use and adoption of secure messaging services. Human end users need to understand how to use the system securely and the effort required to do so must be acceptable for the perceived benefits.

   In previous research, various secure messaging tools have been evaluated and weaknesses in the HCI portion of their design have been revealed. The seminal paper "Why Johnny Can't Encrypt" [WT99] along with follow-up studies evaluating PGP tools [GM05; GMS+05] and other messaging protocols [SDOF07; SYG08; CGM+11; FHM+12; RKB+13] have also showed users encountering severe problems using encryption securely. However, these studies focused on UI issues unique to specific implementations. This approach results in few generic insights regarding secure messenger protocol and application design. Given the huge number of secure messaging implementations and academic approaches considered in our systematization, we opted to extract generic concepts. Because we focus on usability consequences imposed by generic concepts, our results hold for any tool that implements these concepts.

   To evaluate the usability of secure messaging approaches, we examine the additional user effort (and decisions), security-related errors, and reduction in reliability and flexibility that they introduce. Our usability metrics compare this extra effort to a baseline approach with minimal security or privacy features. This is a challenging task and conventional user studies are not well suited to extract such high-level usability comparisons between disparate tools. We opted to employ expert reviews to measure these usability properties, which is consistent with previous systematization

efforts for security schemes in other areas [BHvS12; Cv13]. To consider usability and adoption hurdles in practice, we combined these expert reviews with cognitive walkthroughs of actual implementations based on Nielsen's usability principles [Nie92; Nie94; JM95] and already known end-user issues discovered in previous work [WT99; GM05; GMS+05; SBKH06; SYG08; CGM+11; FHM+12; RVR14]. These usability results supplement our technical systematization and highlight potential trade-offs between security and usability.

3. **Ease of Adoption**: Adoption of secure messaging schemes is not only affected by their usability and security claims, but also by requirements imposed by the underlying technology. Protocols might introduce adoption issues by requiring additional resources or infrastructure from end users or service operators. When evaluating the adoption properties of an approach, we award a good score if the system does not exceed the resources or infrastructure requirements of a baseline approach that lacks any security or privacy features.

2.3 Trust Establishment

One of the most challenging aspects of messaging security is *trust establishment*, the process of users verifying that they are actually communicating with the parties they intend. *Long-term key exchange* refers to the process where users send cryptographic key material to each other. *Long-term key authentication* (also called *key validation* and *key verification*) is the mechanism allowing users to ensure that cryptographic long-term keys are associated with the correct real-world entities. We use *trust establishment* to refer to the combination of *long-term key exchange* and *long-term key authentication* in the remainder of this thesis. After contact discovery (the process of locating contact details for friends using the messaging service), end users first have to perform trust establishment in order to enable secure communication. In Table 2.1, we compare the features of existing trust establishment approaches.

2.3.1 Security and Privacy Features

We identified the following security and privacy features for trust establishment protocols:

- **Network MitM Prevention**: Prevents Man-in-the-Middle (MitM) attacks by local and global network adversaries.

Scheme	Example	Security Features						Usability									Adoption					
		Network MitM Prevented	Operator MitM Prevented	Operator MitM Detected	Key Revocation Possible	Operator Accountability	Privacy Preserving	Automatic Key Initialization	Low Key Maintenance	Easy Key Discovery	Easy Key Recovery	No Shared Secrets	Alert-less Key Renewal	In-Band	Inattentive User Resistant	Immediate Enrollment	Multiple Key Support	No Service Provider	No Auditing Required	No Name Squatting	Asynchronous	Scalable
Opportunistic Encryption†*	TCPCrypt	-	-	-	-	-	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●
+TOFU (Strict)†	-	◐	◐	◐	◐	-	●	●	●	●	-	●	●	-	●	●	●	-	●	●	●	●
+TOFU†*	TextSecure	◐	◐	◐	◐	-	●	●	●	●	●	●	●	-	●	-	-	-	●	●	●	●
Key Fingerprint Verification†*	Threema	●	●	●	●	◐	●	-	-	-	-	-	●	-	-	-	-	-	●	●	●	●
+Short Auth Strings (Out-of-Band)†*	SilentText	●	●	●	●	◐	●	-	-	-	-	-	●	-	-	-	-	-	-	●	●	●
+Short Auth Strings (In-Band/Voice/Video)†*	ZRTP	●	●	●	●	◐	●	-	-	-	-	◐	●	-	-	-	-	-	●	●	●	●
+Socialist Millionaire (SMP)†*	OTR	●	●	●	●	◐	●	-	-	-	-	●	-	-	-	-	-	-	●	●	●	●
+Mandatory Verification†*	SafeSlinger	●	●	●	●	◐	●	-	-	-	-	●	-	-	●	●	-	-	●	●	●	●
Key Directory†*	iMessage	●	-	-	-	●	-	●	●	●	●	●	●	●	●	●	●	●	●	-	●	●
+Certificate Authority†*	S/MIME	●	-	-	-	-	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●
+Transparency Log	-	●	-	◐	◐	-	-	●	●	●	●	●	●	●	●	◐	●	●	●	-	●	●
+Extended Transparency Log†	-	●	-	◐	●	●	-	●	●	●	●	●	●	●	●	◐	●	●	●	-	●	●
+Self-Auditable Log†	CONIKS	●	-	◐	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	◐	●	●
Web-of-Trust†*	PGP	●	●	●	◐	◐	-	-	-	◐	◐	-	-	-	-	-	-	-	●	●	●	●
+Trust Delegation†*	GnuNS	●	●	●	◐	◐	●	-	-	◐	◐	-	-	-	-	-	-	-	●	●	●	●
+Tracking*	Keybase	●	◐	◐	-	◐	-	◐	◐	◐	◐	-	-	-	-	●	-	-	●	-	◐	●
Pure IBC†	SIM-IBC-KMS	●	-	-	-	-	●	●	●	●	●	●	●	●	●	●	●	●	-	-	-	●
+Revocable IBC†	-	●	-	-	-	◐	●	●	●	●	●	●	●	●	●	●	●	●	-	-	-	●
Blockchains*	Namecoin	●	●	●	●	-	●	●	◐	●	-	●	●	-	●	-	●	-	●	●	-	-
Key Directory+TOFU+Optional Verification†*	TextSecure	◐	◐	◐	◐	●	-	●	●	●	●	●	●	●	-	●	-	-	◐	-	●	●
Opportunistic Encryption+SMP†*	OTR	◐	◐	◐	◐	●	●	-	●	-	-	◐	◐	-	-	●	-	-	●	●	●	●

● = provides property; ◐ = partially provides property; - = does not provide property;
†has academic publication; *end-user tool available

Table 2.1: Trade-offs for combinations of trust establishment approaches. Secure approaches often sacrifice usability and adoption.

- **Operator MitM Prevention**: Prevents MitM attacks executed by infrastructure operators.
- **Operator MitM Detection**: Allows the detection of MitM attacks performed by operators after they have occurred.
- **Operator Accountability**: It is possible to verify that operators behaved correctly during trust establishment.
- **Key Revocation Possible**: Users can revoke and renew keys (e.g., to recover from key loss or compromise).
- **Privacy Preserving**: The approach leaks no conversation metadata to other participants or even service operators.

2.3.2 Usability Properties

Most trust establishment schemes require key management: user agents must generate, exchange, and verify other participants' keys. For some approaches, users may be confronted with additional tasks, as well as possible warnings and errors, compared to classic tools without end-to-end security. If a concept requires little user effort and introduces no new error types, we award a mark for the property to denote good usability. We only consider the minimum user interaction required by the protocol instead of rating specific implementations.

- **Automatic Key Initialization**: No additional user effort is required to create a long-term key pair.
- **Low Key Maintenance**: Key maintenance encompasses recurring effort users have to invest into maintaining keys. Some systems require that users sign other keys or renew expired keys. Usable systems require no key maintenance tasks.
- **Easy Key Discovery**: When new contacts are added, no additional effort is needed to retrieve key material.
- **Easy Key Recovery**: When users lose long-term key material, it is easy to revoke old keys and initialize new keys (e.g., simply reinstalling the app or regenerating keys is sufficient).

- **In-band**: No *out-of-band* channels are needed that require users to invest additional effort to establish.
- **No Shared Secrets**: Shared secrets require existing social relationships. This limits the usability of a system, as not all communication partners are able to devise shared secrets.
- **Alert-less Key Renewal**: If other participants renew their long-term keys, a user can proceed without errors or warnings.
- **Immediate Enrollment**: When keys are (re-)initialized, other participants are able to verify and use them immediately.
- **Inattentive User Resistant**: Users do not need to carefully inspect information (e.g., key fingerprints) to achieve security.

2.3.3 Adoption Properties

Trust establishment schemes can exhibit some additional properties that can help them to attain widespread adoption:

- **Multiple Key Support**: Users should not have to invest additional effort if they or their conversation partners use multiple public keys, making the use of multiple devices with separate keys transparent. While it is always possible to share one key on all devices and synchronize the key between them, this can lead to usability problems.
- **No Service Provider Required**: Trust establishment does not require additional infrastructure (e.g., key servers).
- **No Auditing Required**: The approach does not require auditors to verify correct behavior of infrastructure operators.
- **No Name Squatting**: Users can choose their names and can be prevented from reserving a large number of popular names.
- **Asynchronous**: Trust establishment can occur asynchronously without all conversation participants online.
- **Scalable**: Trust establishment is efficient, with resource requirements growing logarithmically (or smaller) with the the total number of participants in the system.

2.3.4 Evaluation

2.3.4.1 Opportunistic Encryption (Baseline)

We consider *opportunistic encryption*, in which an encrypted session is established without any key verification, as a baseline. For instance, this could be an OTR encryption session without any authentication. The main goal of opportunistic encryption is to counter passive adversaries; active attackers can easily execute MitM attacks. From a usability perspective, this approach is the baseline since it neither places any burden on the user nor generates any new error or warning messages.

2.3.4.2 TOFU

Trust-On-First-Use (TOFU) extends opportunistic encryption by remembering previously seen key material [WAP08]. The *network MitM prevented* and *infrastructure MitM prevented* properties are only partially provided due to the requirement that no attacker is present during the initial connection. TOFU *requires no service provider* since keys can be exchanged by the conversation participants directly. TOFU does not define a mechanism for *key revocation*. TOFU can be implemented in strict and non-strict forms. The strict form fails when the key changes, providing *inattentive user resilience* but preventing *easy key recovery*. The non-strict form prompts users to accept key changes, providing *easy key recovery* at the expense of *inattentive user resilience*.

TOFU-based approaches, like the baseline, do not require any user interaction during the initial contact discovery. This yields good scores for all user-effort properties except for the *key revocation* property, which is not defined, and *alert-less key renewal*, since users cannot distinguish benign key changes from MitM attacks without additional verification methods. For instance, TextSecure shows a warning that a user's key has changed and the user must either confirm the new key or apply manual verification to proceed (shown in Figure 2.2). If the user chooses to accept the new key immediately, it is possible to perform the verification later. The motivation behind this approach is to provide more transparency for more experienced or high-risk users, while still offering an "acceptable" solution for novice end users. Critically, previous work in the related domain of TLS warnings has shown that frequent warning messages leads to higher click-through rates in dangerous situations, even with experienced users [SEA+09; AKJ+15].

From a usability and adoption perspective, TOFU performs similarly to the baseline, except for *key recovery* in the strict version and *multiple key support* in both versions. The multiple key support problem arises from the fact that if multiple keys are used, the

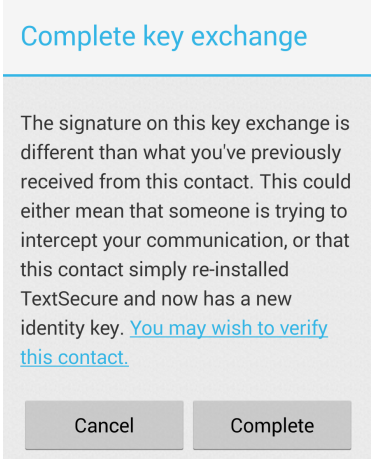


Figure 2.2: TextSecure warning for key changes: the user must either accept the new key by selecting "complete", or perform manual verification [Ope13c].

protocol cannot distinguish between devices. An attacker can claim that a new device, with the attacker's key, is being used.

2.3.4.3 Key Fingerprint Verification

Manual verification requires users to compare some representation of a cryptographic hash of their partners' public keys out of band (e.g., in person or via a separate secure channel).

Assuming the fingerprint check is performed correctly by end users, manual verification provides all desirable security properties with the exception of only partial *key revocation* support, as this requires contacting each communication partner out-of-band. The approaches differ only in their usability and adoption features.

Fingerprint verification approaches introduce severe usability and adoption limitations: users have to perform manual verification before communicating with a new partner (and get them to do the same) to ensure strong authentication. Thus, manual verification does not offer *automatic key initialization*, *easy key discovery*, or *immediate enrollment*. In addition, new keys introduce an alert on key renewal, resulting in a *key maintenance* effort. Fingerprints complicate *multiple key support* since each device might use a different key.

While it is possible to improve the usability of key fingerprint verification by making it optional and combining it with other approaches, we postpone discussion of this strategy until Section 2.3.5.

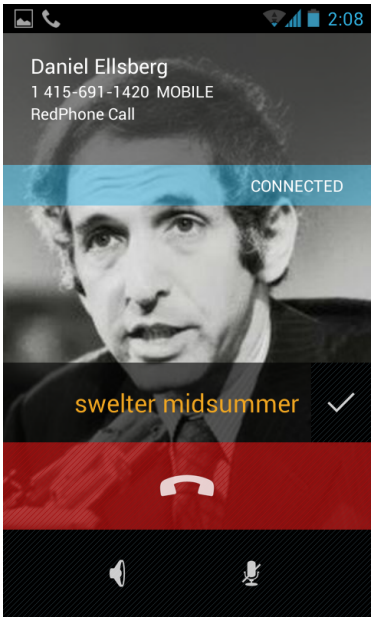


Figure 2.3: Users read random words during SAS verification in RedPhone [Ope13c].

2.3.4.4 Short Authentication Strings

To ease fingerprint verification, shorter strings can be provided to the users for comparison. A short authentication string (SAS) is a truncated cryptographic hash (e.g., 20–30 bits long) of all public parts of the key exchange. It is often represented in a format aimed to be human friendly, such as a short sequence of words. All participants compute the SAS based on the key exchange they observed, and then compare the resulting value with each other. The method used for comparison of the SAS must authenticate the entities using some underlying trust establishment mechanism.

Several encrypted voice channels, including the ZRTP protocol and applications like RedPhone, Signal, and SilentPhone, use the SAS method by requiring participants to read strings aloud [Blo99; ZJC11]. Figure 2.3 shows an example of SAS verification during establishment of a voice channel in RedPhone. For usability reasons, RedPhone and SilentPhone use random dictionary words to represent the hash. Because these tools require the user to end the established call manually if the verification fails, they are not *inattentive user resistant*.

SAS systems based on voice channels anchor trust in the ability of participants to recognize each other's voices. Users who have never heard each other's voices cannot au-

thenticate using this method. Even for users that are familiar with each other, the security provided by voice identification has been the subject of controversy [GS07; PHJ+08]. Recent work [SS14] suggests that, with even a small number of samples of a target user's speaking voice, audio samples can be synthesized that are indistinguishable from the genuine user's voice with typical levels of background noise. We should expect that artificial voice synthesis will improve in cost and accuracy, while human auditory recognition will not improve.

For this reason, we consider voice-based SAS verification to be obsolescent from a security standpoint. In Table 2.1, we assume that users verify the SAS with a method providing stronger security (e.g., using audio and video channels with careful inspection during the SAS verification). If the communication channel (e.g., text messaging) does not support a mechanism to establish trust, the SAS must be compared out of band (e.g., as recommended by SilentText).

The SAS approach sacrifices *asynchronicity*, since mutual authentication must be done with all users at the same time. Due to the short size of the SAS, the naïve approach is vulnerable to a MitM attack by an adversary that attempts to select key exchange values that produce a hash collision for the two connections. To mitigate this problem, the attacker can be limited to a single guess by forcing them to reveal their chosen keys before observing the keys of the honest parties. This can be accomplished by requiring that the initiator of the key exchange release a commitment to their key, and then open the commitment after the other party reveals theirs.

2.3.4.5 Secret-based Zero-Knowledge Verification

The Socialist Millionaire Protocol (SMP) is a zero-knowledge proof of knowledge protocol that determines if secret values held by two parties are equal without revealing the value itself. This protocol is used in OTR as the recommended method for user verification [JY96; AG07]. Alice poses a question based on shared knowledge to Bob *in-band* and secretly records her answer. After Bob answers the question, the two parties perform the SMP to determine if their answers match, without revealing any additional information. Users are expected to choose secure questions with answers based on shared knowledge that attackers would be unable to know or guess.

The SMP used in OTR is performed on a cryptographic hash of the session identifier, the two parties' fingerprints, and their secret answers. This prevents MitM and replay attacks.

Since a MitM must perform an online attack and can only guess once, even low min-entropy secrets achieve strong security [BST01; AG07]. However, use of the SMP sacrifices *asynchronicity* since all participants must be online during the verification. If the protocol fails, the end users do not know whether their answers did not match, or if a MitM attacker exists and has made an incorrect guess.

2.3.4.6 Mandatory Verification

The previously defined verification methods are prone to *inattentive users*. *Mandatory verification* approaches counter user negligence by requiring that users enter the correct fingerprint strings instead of merely confirming that they are correct. Of course, entering the fingerprints takes user effort. In practice, QR-Codes and NFC are popular methods to ease this process.

In SafeSlinger the user must choose the correct answer among three possibilities to proceed [FLK+13]. Physically co-located users form a group and exchange ephemeral keys. Each device hashes all received information and displays the hash as a sequence of three common words. Two additional sequences are randomly generated. The users communicate to determine the sequence that is common to all devices and select it to verify the ephemeral keys, preventing users from simply clicking an "OK" button. These keys are then used to exchange contact information within the group with security guarantees including confidentiality and authenticity.

Mandatory verification is a technique that is applied to another trust establishment scheme; the resulting approach inherits the usability properties of the underlying scheme. Incorporating mandatory verification sacrifices *asynchronicity* to ensure *inattentive user resistance*.

2.3.4.7 Authority-Based Trust

In authority-based trust schemes, public keys must be vouched for by one or more trusted authorities.

During key initialization, authorities can verify ownership of public keys by the claimed subjects through various means, such as password-based authentication or email validation. The authority then asserts the keys' validity to other users. Two well-known examples of authority-based trust establishment are public-key directories and certificate authority schemes.

A Certificate Authority (CA) may issue signed certificates of public keys to users, who can then present them directly to other users without needing to communicate further with the authority. This model has been widely deployed on the web with the X.509 Public Key Infrastructure (PKIX) for HTTPS. While the S/MIME standard uses this model for secure email, it has seen less widespread deployment than PGP.

Alternatively, users may look up keys directly from an online *public-key directory* over a secure channel. This is common in several proprietary messaging applications such as Apple iMessage and BlackBerry Protected Messenger. In contrast to CA schemes, where the conversation partner directly provides an ownership assertion from the CA, the authority is directly asked for ownership assertions in key directory schemes.

From the security point of view, the two schemes only differ in *key revocation* and *privacy preservation*. While key updates in key directories imply the revocation of old keys, in the CA approach, certificates signed by the authority are trusted by default; revocation lists have to be maintained separately. However, CA-based revocation lists used in web browsers are known to have issues with effectiveness and practicality [Mar09; GIJ+12; Cv13]. Since certificates may be exchanged by peers directly, the CA-based approach can be *privacy preserving*.

With either system, users are vulnerable to MitM attacks by the authority, which can vouch for, or be coerced to vouch for, false keys. This weakness has been highlighted by recent CA scandals [VAS11; Lan13a]. Both schemes can also be attacked if the authority does not verify keys before vouching for them. Authorities in messaging services often rely on insecure SMS or email verification, enabling potential attacks.

The two approaches both support good usability. Well-known systems using public-key directories, such as iMessage, work without any user involvement.

2.3.4.8 Transparency Logs

A major issue with trusted authorities is that they can vouch for fraudulent keys in an attack. The *Certificate Transparency* protocol [LLK13] requires that all issued web certificates are included in a public log.

This append-only log is implemented using a signed Merkle tree with continual proofs of consistency [LLK13]. Certificates are only trusted if they include cryptographic proof that they are present in the log. This ensures that any keys the authority vouches for will be visible in the log and evidence will exist that the authority signed keys used in an attack.

Certificate Transparency is a specific proposal for logging PKIX certificates for TLS, but the general idea can be applied to authority-based trust establishment in secure messaging. We refer to the general concept as *transparency logs* for the remainder of the chapter. While there are no known deployments to date, Google plans to adapt *transparency logs* for user keys in End-to-End, its upcoming email encryption tool [Goo14a]. In the absence of a concrete definition, we evaluate *transparency logs* based on the certificate transparency protocol.

The main security improvement of the two schemes consists of *operator accountability* and the *detection of operator MitM attacks* after the fact. The remaining security features are inherited from authority-based trust systems.

However, these schemes introduce new and unresolved usability and adoption issues. For instance, the logs must be audited to ensure correctness, negating the *no auditing required* property. The auditing services require gossip protocols to synchronize the view between the monitors and prevent attack bubbles (e.g., where different views are presented to different geographical regions) [LLK13]. Also, since only identity owners are in a position to verify the correctness of their long-term keys, they share responsibility for verifying correct behavior of the log. Previous research has shown that users often neglect such security responsibilities [SEA+09], so this task should be performed automatically by client applications. However, if a client detects a certificate in the log that differs from their version, it is not clear whether the authorities have performed an attack, an adversary has successfully impersonated the subject of the certificate to the authorities, or if the subject actually maintains multiple certificates (e.g., due to installing the app on a second device). Ultimately, end users have to cope with additional security warnings and errors, and it remains to be seen whether they can distinguish between benign and malicious log discrepancies without training. In addition, *transparency logs* might hamper *immediate enrollment* due to delays in log distribution.

Ryan [Rya14] proposed extending the *transparency logs* concept using two logs: one of all certificates in chronological order of issuance, and one of currently valid certificates sorted lexicographically. This enables a form of revocation by making it efficient to query which certificates are currently valid for a given username.

Melara et al. [MBB+14] proposed CONIKS, using a series of chained commitments to Merkle prefix trees to build a key directory that is self-auditing, that is, for which individual users can efficiently verify the consistency of their own entry in the directory without relying on a third party. This "self-auditing log" approach makes the system partially have *no auditing required* (as general auditing of non-equivocation is still required) and also enables the system to be *privacy preserving* as the entries in the directory need not be made public.

This comes at a mild bandwidth cost not reflected in our table, estimated to be about 10 kilobytes per client per day for self-auditing.

Both Ryan's Extended Certificate Transparency and CONIKS also support a *proof-of-absence*, which guarantees the absence of an identifier or key in the log.

2.3.4.9 Web of Trust

In a *web of trust* scheme, users verify each other's keys using manual verification and, once they are satisfied that a public key is truly owned by its claimed owner, they sign the key to certify this. These certification signatures might be uploaded to key servers. If Alice has verified Bob's key, and Bob certifies that he has verified Carol's key, Alice can then choose to trust Carol's key based on this assertion from Bob. Ideally, Alice will have multiple certification paths to Carol's key to increase her confidence in the key's authenticity. Zimmermann introduced the concept of a web-of-trust scheme as part of PGP [Zim95].

The user interface for web of trust schemes tends to be relatively complex and has never been fully standardized. The scheme also requires a well-connected social graph, hence the motivation for "key-signing parties" to encourage users to form many links within a common social context.

Assuming that the web of trust model performs correctly, MitM attacks by network and operator adversaries are limited due to distribution of trust. However, since key revocations and new keys might be withheld by key servers, the model offers only partial *operator accountability* and *key revocation*. Since the web of trust model produces a public social graph, it is not *privacy preserving*.

The key-initialization phase requires users to get their keys signed by other keys, so the system does not offer *automatic key initialization*, *alert-less key renewal*, or *immediate enrollment*, and is not *inattentive user resistant*. Because users must participate in key-signing parties to create many paths for trust establishment, users have a high *key maintenance* overhead and a need for an *out-of-band* channel. Even worse, users must understand the details of the PKI and be able to decide whether to trust a key.

PGP typically uses a web of trust for email encryption and signing. In practice, the PGP web of trust consists of one strongly connected component and many unsigned keys or small connected components, making it difficult for those outside the strongly connected component to verify keys [UHHC11].

A simplification of the general web of trust framework is SDSI [RL96] (Simple Distributed Security Infrastructure) later standardized as SPKI [Ell96; EFL+99] (Simple

Public Key Infrastructure). With SDSI/SPKI, Bob can assert that a certain key belongs to "Carol" and, if Alice has verified Bob's key as belonging to "Bob", that key will be displayed to Alice as "Bob's Carol" until Alice manually verifies Carol's key herself (which she can then give any name she wants, such as "Carol N."). We refer to these approaches as *trust delegation*. A modern implementation is the GNU Name System (GNS) [WSG14a; WSG14b], which implements SDSI/SPKI-like semantics with a key server built using a distributed hash table to *preserve privacy*.

2.3.4.10 Keybase

Keybase is a trust establishment scheme allowing users to find public keys associated with social network accounts [Key14]. It is designed to be easily integrated into other software to provide username-based trust establishment. If a user knows a social network username associated with a potential conversation partner, they can use Keybase to find the partner's public key.

During key initialization, all users register for accounts with the Keybase server. They then upload a public key and proof that they own the associated private key. Next, the user can associate accounts on social networks or other services with their Keybase account. Each external service is used to post a signature proving that the account is bound to the named Keybase account.

When looking up the key associated with a given user, the Keybase server returns the public key, a list of associated accounts, and web addresses for the external proofs. The client software requests the proofs from the external services and verifies the links. The user is then prompted to verify that the key belongs to the expected individual, based on the verified social network usernames. To avoid checking these proofs for every cryptographic operation, the user can sign the set of accounts owned by their partner. This signature is stored by the Keybase server so that all devices owned by the user can avoid verifying the external proofs again. This process is known as *tracking*. Tracking signatures created by other users are also accessible, providing evidence of account age. Old tracking signatures provide confidence that a user's accounts have not been recently compromised, but does not protect against infrastructure operator attacks.

Keybase provides partial *operator MitM protection* since attacks require collusion between multiple operators. The scheme also provides easier *key initialization* and *key maintenance* than web-of-trust methods.

2.3.4.11 Identity-Based Cryptography

In identity-based cryptography (IBC), first proposed by Shamir [Sha85], plaintext identifiers (such as email or IP addresses) are mapped to public keys. A trusted third party, the Private Key Generator (PKG), publishes a PKG public key that is distributed to all users of the system. Public keys for an identifier are computed using a combination of the identifier and the PKG public key. The owner of the identity requests the private key for that identity from the PKG while providing proof that they own the identity. The advantage of this system is that users do not need to contact any other entity in order to retrieve the public key of a target user, since the public key is derived from the identifier.

There are two main problems with basic IBC schemes: they lack any *operator MitM prevention* and *key revocation* is not possible. Since the PKG can generate private keys for any user, the operator of the PKG can break the security properties of all conversations. While this fundamental problem cannot be overcome without using hybrid encryption schemes, key revocation support can be added. Revocable IBC approaches [BGK08; LV09; WLXZ14] add timestamps to the public key derivation process, regularly refreshing key material.

IBC schemes are normally deployed in situations where the trustworthiness of the PKG operator is assumed, such as in enterprise settings. Few pure-IBC schemes have been proposed for end-user messaging [TZ05; BMHD08].

2.3.4.12 Blockchains

The Bitcoin cryptocurrency utilizes a novel distributed consensus mechanism using pseudonymous "miners" to maintain an append-only log [Nak08]. Voting power is distributed in proportion to computational resources by using a probabilistic proof-of-work puzzle. For the currency application, this log records every transaction to prevent double-spending. Miners are rewarded (and incentivized to behave honestly) by receiving money in proportion to the amount of computation they have performed. The success of Bitcoin's consensus protocol has led to enthusiasm that similar approaches could maintain global consensus on other types of data, such as a mapping of human-readable usernames to keys.

Namecoin, the first fork of Bitcoin, allows users to claim identifiers, add arbitrary data (e.g., public keys) as records for those identifiers, and even sell control of their identifiers to others [Nam11]. Namecoin and similar name-mapping blockchains are denoted by the *blockchain* entry in Table 2.1. Unlike most other schemes, Namecoin is strictly "first-come, first-served", with any user able to purchase ownership of any number of unclaimed names

for a small, fixed fee per name. This price is paid in Namecoins—units of currency that are an inherent part of the system. A small maintenance fee is required to maintain control of names, and small fees may be charged by miners to update data or transfer ownership of names.

From the security perspective, blockchain schemes achieve similar results to manual verification, except that instead of exchanging keys, the trust relies on the username only. Once users have securely exchanged usernames, they can reliably fetch the correct keys.

However, various shortcomings arise from a usability and adoption perspective. The primary usability limitation is that if users ever lose the private key used to register their name (which is not the same as the communication key bound to that name), they will permanently lose control over that name (i.e., *key recovery* is not possible). Similarly, if the key is compromised, the name can be permanently and irrevocably hijacked. Thus, the system requires significant *key management* effort and burdens users with high responsibility. If users rely on a web-based service to manage private keys for them, as many do with Bitcoin in practice, the system is no longer truly end-to-end. The system requires users to pay to reserve and maintain names, sacrificing *low key maintenance* and *automatic key initialization*. Users also cannot instantly issue new keys for their identifiers (i.e., there is no *immediate enrollment*) but are required to wait for a new block to be published and confirmed. In practice, this can take 10–60 minutes depending on the desired security level.

On the adoption side, for the system to be completely trustless, users must store the entire blockchain locally and track its progress. Experience from Bitcoin shows that the vast majority of users will not do this due to the communication and storage requirements and will instead trust some other party to track the blockchain for them. This trusted party cannot easily insert spurious records, but can provide stale information without detection. In any case, the system is not highly *scalable* since the required amount of storage and traffic consumption increases linearly with the number of users.

Finally, there are serious issues with *name squatting*, which have plagued early attempts to use the system. Because anybody can register as many names as they can afford, a number of squatters have preemptively claimed short and common names. Given the decentralized nature of blockchains, this is hard to address without raising the registration fees, which increases the burden on all users of the system.

2.3.5 Discussion

As Table 2.1 makes evident, no trust establishment approach is perfect. While it is common knowledge that usability and security are often at odds, our results show exactly where

the trade-offs lie. Approaches either sacrifice security and provide a nearly ideal user experience, or sacrifice user experience to achieve nearly ideal security scores. Authority-based trust (whether in the form of a single authority or multiple providers) and TOFU schemes are the most usable and well-adopted, but only offer basic security properties. Not surprisingly, authority-based trust (particularly app-specific key directories) is predominant among recently developed apps in the wild, as well as among apps with the largest userbases (e.g., iMessage, BlackBerry Protected, TextSecure, and Wickr). By contrast, no approach requiring specific user action to manage keys, such as web-of-trust, Keybase, GNS, or blockchains, has seen significant adoption among non-technically-minded users.

In practice, we may be faced with the constraint that *none* of the usability properties can be sacrificed in a system that will achieve mass adoption. Higher-security schemes may be useful within organizations or niche communities, but defending against mass surveillance requires a communication system that virtually all users can successfully use. Thus, it may be wise to start from the basic user experience of today's widely deployed communication apps and try to add as much security as possible, rather than start from a desired security level and attempt to make it as simple to use as possible. The recent partnership between WhatsApp and TextSecure [Ope14a] exemplifies this approach.

There appears to be considerable room for security improvements over authoritative key directories even without changes to the user experience. Transparency logs might provide more accountability with no interaction from most users. Because this approach has not yet been deployed, it remains to be seen how much security is gained in practice. The insertion of new keys in the log does not provide public evidence of malicious behavior if insecure user authentication methods (e.g., passwords) are used to authorize key changes, as we fully expect will be the case. Still, the possible loss of reputation may be enough to keep the server honest.

Another promising strategy is a layered design, with basic security provided by a central key directory, additional trust establishment methods for more experienced users (e.g., visual fingerprint verification or QR-codes), and TOFU warning messages whenever contacts' keys have changed. TextSecure and Threema, among others, take such a layered approach (represented by the second-to-last row in Table 2.1). In contrast, OTR uses opportunistic encryption with the ability to perform the SMP to ensure trust (represented by the last row in Table 2.1).

Conversely, the approaches with good security properties should focus on improving usability. There has been little academic work studying the usability of trust establishment. Further research focusing on end-users' mental models and perception for trust establishment could help to develop more sophisticated and understandable approaches.

2.4 Conversation Security

After *trust establishment* has been achieved, a *conversation security* protocol protects the security and privacy of the exchanged messages. This encompasses how messages are encrypted, the data and metadata that messages contain, and what cryptographic protocols (e.g., ephemeral key exchanges) are performed. A conversation security scheme does not specify a trust establishment scheme nor define how transmitted data reaches the recipient.

In Table 2.2, we compare the features of existing approaches for conversation security. Rows without values in the "group chat" columns can only be used in a two-party setting.

2.4.1 Security and Privacy Features

- **Confidentiality**: Only the intended recipients are able to read a message. Specifically, the message must not be readable by a server operator that is not a conversation participant.
- **Integrity**: No honest party will accept a message that has been modified in transit.
- **Authentication**: Each participant in the conversation receives proof of possession of a known long-term secret from all other participants that they believe to be participating in the conversation.[1] In addition, each participant is able to verify that a message was sent from the claimed source.
- **Participant Consistency**: At any point when a message is accepted by an honest party, all honest parties are guaranteed to have the same view of the participant list.
- **Destination Validation**: When a message is accepted by an honest party, they can verify that they were included in the set of intended recipients for the message.
- **Forward Secrecy**: Compromising all key material does not enable decryption of previously encrypted data.
- **Backward Secrecy**: Compromising all key material does not enable decryption of succeeding encrypted data.
- **Anonymity Preserving**: Any anonymity features provided by the underlying transport privacy architecture are not undermined (e.g., if the transport privacy system

[1] We define authentication in this manner in order to decouple it from *participant consistency*.

Scheme	Example	Security and Privacy															Adoption					Group Chat				
		Confidentiality	Integrity	Authentication	Participant Consistency	Destination Validation	Forward Secrecy	Backward Secrecy	Anonymity Preserving	Speaker Consistency	Causality Preserving	Message Unlinkability	Global Transcript	Message Unlinkability	Particip. Repudiation	Message Repudiation	Out-of-Order Resilient	Dropped Message Resilient	Asynchronicity	Multi-Device Support	No Additional Service	Computational Equality	Trust Equality	Subgroup Messaging	Contractable	Expandable
TLS+Trusted Server†*	Skype	-	-	-	-	-	-	-	-	-	-	-	-	●	●	●	●	●	●	●	-	●	●	●	◐	◐
Static Asymmetric Crypto†*	OpenPGP	●	●	●	-	-	-	-	●	-	-	-	-	-	-	-	●	●	●	●	●	●				
+IBE†	Wang et al.	-	●	●	-	-	-	-	●	-	-	-	-	-	-	-	●	●	●	●	-					
+Short Lifetime Keys	OpenPGP Draft	●	●	●	-	-	◐	◐	●	-	-	-	-	-	-	-	●	●	●	●	-					
+Non-Interactive IBE†	Canetti et al.	●	●	●	-	-	●	-	●	-	-	-	-	-	-	-	◐	●	●	●	●					
+Puncturable Encryption†	Green and Miers	●	●	●	-	-	●	-	●	-	-	-	-	-	-	-	●	●	●	●	●					
Key Directory+Short Lifetime Keys†	IMKE	●	●	◐	-	●	◐	◐	-	-	-	-	●	●	●		●	●	-	-	-					
+Long-Term Keys†	SIMPP	●	●	◐	-	●	◐	◐	-	-	-	-	●	●	-		●	●	-	-	-					
Authenticated DH†*	TLS-EDH-MA	●	●	●	●	●	◐	◐	●	-	-	-	●	●	◐		●	●	-	-	●					
+Naïve KDF Ratchet*	SCIMP	●	●	●	●	●	●	◐	●	◐	-	-	●	●	◐		◐	◐	-	-	●					
+DH Ratchet†*	OTR	●	●	●	●	●	◐	●	●	◐	◐	-	●	●	◐		◐	◐	-	-	●					
+Double Ratchet†*	Axolotl	●	●	●	●	●	●	●	●	◐	◐	-	●	●	◐		●	◐	-	-	●					
+Double Ratchet+3DH AKE†*	-	●	●	●	●	●	●	●	◐	◐	◐	-	●	●	●		●	◐	-	-	●					
+Double Ratchet+3DH AKE+Prekeys†*	TextSecure	●	●	●	●	●	●	●	-	◐	●	-	●	●	●		◐	◐	●	-	-					
Key Directory+Static DH+Key Transport†	Kikuchi et al.	●	●	-	-	●	◐	◐	-	-	-	-	●	●	-		●	●	●	-	-	-	-	-	●	●
+Authenticated EDH+Group MAC†	GROK	●	●	◐	-	●	◐	◐	●	-	-	-	●	●	-		●	●	●	-	-	-	-	-	●	●
GKA+Signed Messages+Parent IDs†	OldBlue	●	●	●	●	●	◐	◐	●	●	-	-	-	-	-		●	●	◐	-	●	●	●	-	-	-
Authenticated MP DH+Causal Blocks†*	KleeQ	●	●	◐	◐	◐	●	●	●	◐	◐	●	-	●	●		●	●	◐	-	●	●	●	-	-	●
OTR Network+Star Topology†	GOTR (2007)	●	●	-	-	-	◐	●	-	-	-	-	●	●	●		◐	●	◐	-	●	-	-	-	●	●
+Pairwise Topology†		●	●	●	●	●	◐	●	●	-	-	-	●	●	●		◐	●	◐	-	●	●	●	●	●	●
+Pairwise Axolotl+Multicast Encryption*	TextSecure	●	●	●	-	●	●	●	-	●	●	-	●	●	●		●	●	●	-	-	●	●	●	●	●
DGKE+Shutdown Consistency Check†	mpOTR	●	●	●	●	●	◐	◐	●	◐	-	-	-	●	●		●	●	-	-	●	●	●	-	-	-
Circle Keys+Message Consistency Check†	GOTR (2013)	●	●	●	●	●	◐	◐	●	●	●	●	●	●	●		●	-	-	-	◐	●	●	-	●	●

● = provides property; ◐ = partially provides property; - = does not provide property;
†has academic publication; *end-user tool available

Table 2.2: Conversation security protocols and their usability and adoption implications. No approach requires additional user effort.

provides anonymity, the conversation security level does not deanonymize users by linking key identifiers).

- **Speaker Consistency**: All participants agree on the sequence of messages sent by each participant. A protocol might perform consistency checks on blocks of messages during the protocol, or after every message is sent.
- **Causality Preserving**: Implementations can avoid displaying a message before messages that causally precede it.
- **Global Transcript**: All participants see all messages in the same order.

Not all security and privacy features are completely independent. If a protocol does not authenticate participants, then it offers participation repudiation (since no proof of participation is ever provided to anyone). Similarly, no authentication of message origin implies message repudiation as well as message unlinkability. Note that the implications are only one way: repudiation properties might be achieved together with authentication. Additionally, a global transcript order implies both speaker consistency and causality preservation since all transcripts are identical.

Conversation security schemes may provide several different forms of deniability. Based on the definitions from Section 3.1, we define the following deniability-related features:

- **Message Unlinkability**: If a judge is convinced that a participant authored one message in the conversation, this does not provide evidence that they authored other messages.
- **Message Repudiation**: Given a conversation transcript and all cryptographic keys, there is no evidence that a given message was authored by any particular user. We assume that the accuser has access to the session keys because it is trivial to deny writing a plaintext message when the accuser cannot demonstrate that the ciphertext corresponds to this plaintext. We also assume that the accuser does not have access to the accused participant's long-term secret keys because then it is simple for the accuser to forge the transcript (and thus any messages are repudiable).
- **Participation Repudiation**: Given a conversation transcript and all cryptographic key material for all but one accused (honest) participant, there is no evidence that the honest participant was in a conversation with any of the other participants.

2.4.2 Usability and Adoption

In classic messaging tools, users must only reason about two simple tasks: sending and receiving messages. However, in secure communication, additional tasks might be added. In old secure messaging systems, often based on OpenPGP, users could manually decide whether to encrypt and/or sign messages. Many studies have shown that this caused usability problems [WT99; GM05; GMS+05; FHM+12; RVR14]. However, during our evaluation, we found that most recent secure messenger apps secure all messages by default without user interaction. Since all implementations can operate securely once the trust establishment is complete, we omit the user-effort columns in Table 2.2. However, we take other usability and adoption factors, such as resilience properties, into account:

- **Out-of-Order Resilient**: If a message is delayed in transit, but eventually arrives, its contents are accessible upon arrival.
- **Dropped Message Resilient**: Messages can be decrypted without receipt of all previous messages. This is desirable for asynchronous and unreliable network services.
- **Asynchronous**: Messages can be sent securely to disconnected recipients and received upon their next connection.
- **Multi-Device Support**: A user can participate in the conversation using multiple devices at once. Each device must be able to send and receive messages. Ideally, all devices have identical views of the conversation. The devices might use a synchronized long-term key or distinct keys.
- **No Additional Service**: The protocol does not require any infrastructure other than the protocol participants. Specifically, the protocol must not require additional servers for relaying messages or storing any kind of key material.

2.4.3 Group Chat Features

Several additional features are only meaningful for group protocols (i.e., protocols supporting chats between three or more participants):

- **Computational Equality**: All chat participants share an equal computational load.
- **Trust Equality**: No participant is more trusted or takes on more responsibility than any other.

- **Subgroup messaging**: Messages can be sent to a subset of participants without forming a new conversation.
- **Contractible Membership**: After the conversation begins, participants can leave without restarting the protocol.
- **Expandable Membership**: After the conversation begins, participants can join without restarting the protocol.

When a participant joins a secure group conversation, it is desirable for the protocol to compute new cryptographic keys so that the participant cannot decrypt previously sent messages. Likewise, keys should be changed when a participant leaves so that they cannot read new messages. This is trivial to implement by simply restarting the protocol, but this approach is often computationally expensive. Protocols with *expandable / contractible membership* achieve this without restarts.

There are many higher-level security and privacy design issues for secure group chat protocols. For example, the mechanisms for inviting participants to chats, kicking users out of sessions, and chat room moderation are all important choices that are influenced by the intended use cases. We do not cover these features here because they are implemented at a higher level than the secure messaging protocol layer.

2.4.4 Two-party Chat Evaluation

2.4.4.1 Trusted Central Servers (Baseline)

The most basic conversation security features that a secure chat protocol can provide are confidentiality and integrity. This can be easily implemented without adversely affecting usability and adoption properties by using a central server to relay messages and securing connections from clients to the central server using a transport-layer protocol like TLS. This also allows the central server to provide presence information. Since this approach does not negatively affect usability, it is no surprise that this architecture has been adopted by some of the most popular messaging systems today (e.g., Skype, Facebook Chat, Google Hangouts) [Sai11; SFK+12; Mic14; Goo14b; Fac14]. We do not consider these protocols further because they allow the central server to decrypt messages and thus do not meet our stronger end-to-end definition of *confidentiality*—that messages cannot be read by anyone except the intended recipient(s). We include this approach as a baseline in Table 2.2 in order to evaluate the effects of various designs.

Note that the baseline protocols provide all *repudiation* features, since there is no cryptographic proof of any activity. Additionally, these protocols are highly resilient to errors since there are no cryptographic mechanisms that could cause problems when messages are lost. The use of a trusted central server makes *asynchronicity* and *multi-device support* trivial.

2.4.4.2 Static Asymmetric Cryptography

Another simple approach is to use participants' static long-term asymmetric keypairs for signing and encrypting.

OpenPGP and S/MIME are two well-known and widely implemented standards for message protection, mostly used for email but also in XMPP-based tools [CDF+99; FL04; RT10; Sai11].

While this approach provides *confidentiality*, message *authentication*, and *integrity*, it causes a loss of all forms of *repudiation*. Additionally, care must be taken to ensure that *destination validation* and *participant consistency* checks are performed. Without destination validation, *surreptitious forwarding* attacks are possible [Dav01]. Without participant consistency, *identity misbinding* attacks might be possible [DvW92]. Defenses against replay attacks should also be included. These considerations are particularly relevant since the OpenPGP and S/MIME standards do not specify how to provide these features, and thus most implementations remain vulnerable to all of these attacks [CDF+99; RT10].

To simplify key distribution, several authors have proposed the use of identity-based cryptography in the same setting. The SIM-IBC-KMS protocol acts as an overlay on the MSN chat network with a third-party server acting as the PKG [BMHD08]. Messages are encrypted directly using identity-based encryption. The protocol from Wang et al. [WLL13] operates similarly, but distributes the PKG function across many servers with a non-collusion assumption in order to limit the impact of a malicious PKG. These protocols partially sacrifice *confidentiality* since an attacker with access to the PKG private key could surreptitiously decrypt communications.

A second issue with naïve asymmetric cryptography is the lack of *forward* or *backward secrecy*. One way to address this issue is to use keys with very short lifetimes (e.g., changing the key every day). Brown et al. [BBL02] propose several extensions to OpenPGP based on this principle. In the most extreme proposal, conversations are started using long-term keys, but each message includes an ephemeral public key to be used for replies. This method provides *forward* and *backward secrecy* for all messages except those used to start a conversation.

From a usability and adoption perspective, static key approaches achieve the same properties as the baseline. Apart from the non-transparent trust establishment, iMessage is a prominent example of how static asymmetric cryptography can achieve end-to-end conversation security with no changes to the user experience. Since the same long-term keys are used for all messages, *message order resilience*, *dropped message resilience*, *asynchronicity*, and *multi-device-support* are provided. *No additional services* are required.

2.4.4.3 FS-IBE

In traditional PKI cryptography, *forward secrecy* is achieved by exchanging ephemeral session keys or by changing keypairs frequently. The use of key agreement protocols makes *asynchronicity* difficult, whereas frequently changing keypairs requires expensive key distribution. Forward Secure Identity Based Encryption (FS-IBE) allows keypairs to be changed frequently with a low distribution cost. Unlike traditional identity-based encryption schemes, the private key generators (PKG) in FS-IBE are operated by the end users and not by a server. Initially, each participant generates a PKG for an identity-based cryptosystem. Participants generate N private keys (SK_i), one for each time period i, by using their PKG, and then immediately destroy the PKG. Each private key SK_i is stored encrypted by the previous private key SK_{i-1} [And97; CHK03]. The participant then distributes the public key of the PKG. Messages sent to the participant are encrypted for the private key corresponding to the current time period. When a time period concludes, the next secret key is decrypted and the expired key is deleted. Thus, if intermediate keys are compromised, the attacker can only retrieve corresponding future private keys; *forward secrecy*, but not *backward secrecy*, is provided. In contrast to generating key pairs for each time period, which requires distribution of N keys, only a single public master key is published; however, the generation still needs to be repeated after all time periods expire.

Canetti, Halevi, and Katz were the first to construct a non-interactive forward secrecy scheme based on hierarchical IBE with logarithmic generation and storage costs [CHK03]. In addition, they showed how their scheme can be extended to an unbounded number of periods (i.e., the private keys do not have to be generated in advance), removing the need for *additional services* to distribute new keys at the cost of increasing computational requirements over time. This scheme provides non-interactive *asynchronous forward secrecy* without relying on *additional services*. However, if messages arrive out of order, their corresponding private keys might have already been deleted. As a mitigation, expired keys might be briefly retained, providing partial *out-of-order resilience*.

Green and Miers proposed *puncturable encryption* [GM15], a modification of attribute-based encryption [SW05] in which each message is encrypted with a randomly chosen "tag" and the recipient can update their private key to no longer be able to decrypt messages with that tag after receipt. This approach provides arbitrary *out-of-order resilience*, although to make the scheme efficient in practice requires periodically changing keys.

Computational costs and storage costs increase over time for both FS-IBE and puncturable encryption, introducing *scalability* concerns. To our knowledge, neither approach has been deployed and they thus merit further development.

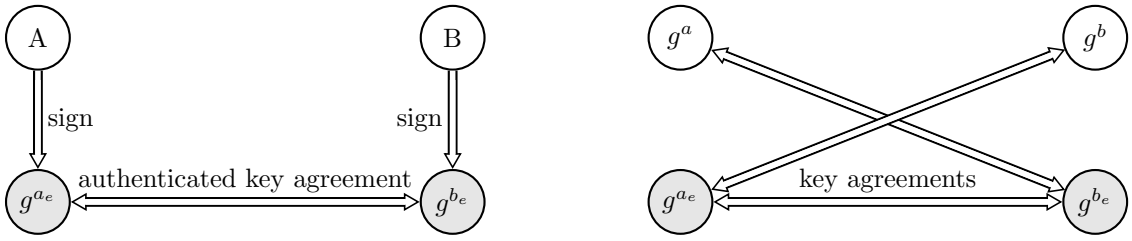
2.4.4.4 Short Lifetime Key Directories

Several protocols make use of a central server for facilitating chat session establishment. In these systems, users authenticate to the central server and upload public keys with short lifetimes. The server acts as a key directory for these ephemeral public keys. Conversations are initiated by performing key exchanges authenticated with the short-term keys vouched for by the key directory. Messages are then encrypted and authenticated using a MAC. IMKE [Mv06] is a protocol of this type where the server authenticates users through the use of a password. SIMPP [YK07; YKAL08; LY09] operates similarly, but uses long-term keys to authenticate instead.

These protocols achieve *confidentiality* and *integrity*, but lack *authentication* of participants since the central server can vouch for malicious short-term keys. Since session keys are exchanged on a per-conversation basis, these protocols achieve *forward* and *backward secrecy* between conversations. Since SIMPP uses signatures during the login procedure, it loses *participation repudiability*; the accuser cannot forge their response to the server's challenge.

2.4.4.5 Authenticated Diffie-Hellman

While the use of central servers for presence information and central authentication is fundamental to systems such as IMKE and SIMPP, there is an alternative class of solutions that instead performs end-to-end authenticated Diffie-Hellman (DH) key exchanges. By default, the authenticated DH key agreement does not rely on central servers. In an authenticated key exchange (AKE) such as authenticated DH, the participants generate an ephemeral session key and authenticate the exchange using their long-term keys. The resulting session key is used to derive symmetric encryption and MAC keys, which then



(a) OTRv1 DH handshake. The session key is derived from the key agreement based on signed ephemeral keys: $s = \mathrm{DH}(g^{a_e}, g^{b_e})$

(b) 3-DH handshake. The session key is a combination of all key agreements: $s = \mathrm{KDF}(\mathrm{DH}(g^{a_e}, g^{b_e})||\mathrm{DH}(g^{a_e}, g^b)||\mathrm{DH}(g^a, g^{b_e}))$

Figure 2.4: TLS/OTRv1 handshake vs. 3-DH handshake (figures derived from [Ope13d]). Gray nodes represent ephemeral keys, white nodes represent long-term keys.

protect messages using an encrypt-then-MAC approach. This basic design provides *confidentiality*, *integrity*, and *authentication*. TLS with an ephemeral DH cipher suite and mutual authentication (TLS-EDH-MA) is a well-known example of this approach. Note that further protections are required during key exchange to protect against *identity misbinding* attacks violating *participant consistency* [DvW92; AG07], such as those provided by SIGMA protocols [Kra03].

The use of ephemeral session keys provides *forward* and *backward secrecy* between conversations. *Message unlinkability* and *message repudiation* are provided since messages are authenticated with shared MAC keys rather than being signed with long-term keys. At a minimum, messages can be forged by any chat participants. Some protocols, such as OTR, take additional measures, such as publication of MAC keys and the use of malleable encryption, to expand the set of possible message forgers [BGB04]. If the participants simply sign all AKE parameters, then this approach does not provide *participation repudiation*. However, if participants only sign their own ephemeral keys, these signatures can be reused by their conversation partners in forged transcripts. Figure 2.4a shows the authenticated key exchange used by OTRv1 (more recent versions use a SIGMA key exchange). Conversation partners are able to reuse ephemeral keys signed by the other party in forged transcripts, thereby providing partial *participation repudiation*. OTR users can increase the number of possible forgers by publishing previously signed ephemeral keys in a public location, thereby improving their *participation repudiation*.

Once the AKE has been performed, the encrypt-then-MAC approach allows messages to be exchanged asynchronously with *out-of-order* and *dropped message resilience*. However, since a traditional AKE requires a complete handshake before actual messages can be encrypted, this basic approach requires *synchronicity* during conversation initialization.

Additionally, since key agreements can only be performed with connected devices, there is no trivial *multi-device support*.

2.4.4.6 Key Evolution

A desirable property is *forward secrecy* for individual messages rather than for entire conversations. This is especially useful in settings where conversations can last for the lifetime of a device. To achieve this, the session key from the initial key agreement can be evolved over time through the use of a *session key ratchet* [Ope13a]. A simple approach is to use key derivation functions (KDFs) to compute future message keys from past keys. This naïve approach, as used in SCIMP [MBZ12], provides *forward secrecy*. However, it does not provide *backward secrecy* within conversations; if a key is compromised, all future keys can be derived using the KDF. *Speaker consistency* is partially obtained since messages cannot be surreptitiously dropped by an adversary without also dropping all future messages (otherwise, recipients would not be able to decrypt succeeding messages). If messages are dropped or arrive out of order, the recipient will notice since the messages are encrypted with an unexpected key. To handle this, the recipient must store expired keys so that delayed or re-transmitted messages can still be decrypted, leaving a larger window of compromise than necessary. Thus, *out-of-order* and *dropped message resilience* are only partially provided.

2.4.4.7 Diffie-Hellman Ratchet

A different ratcheting approach, introduced by OTR, is to attach new DH contributions to messages [BGB04]. With each sent message, the sender advertises a new DH value. Message keys are then computed from the latest acknowledged DH values. This design introduces *backward secrecy* within conversations since a compromised key will regularly be replaced with new key material. *Causality preservation* is partially achieved since messages implicitly reference their causal predecessors based on which keys they use. The same level of *speaker consistency* as the naïve KDF solution can be provided by adding a per-speaker monotonic counter to messages. A disadvantage of the DH ratchet is that session keys might not be renewed for every message (i.e., *forward secrecy* is only partially provided). Like the KDF-based ratchet, the DH ratchet lacks *out-of-order resilience*; if a message arrives after a newly advertised key is accepted, then the necessary decryption key was already deleted.

2.4.4.8 Double-Ratchet (Axolotl)

To improve the forward secrecy of a DH ratchet, both ratchet approaches can be combined: session keys produced by DH ratchets are used to seed per-speaker KDF ratchets. Messages are then encrypted using keys produced by the KDF ratchets, frequently refreshed by the DH ratchet on message responses. The resulting *double ratchet*, as implemented by Axolotl [Per13], provides *forward secrecy* across messages due to the KDF ratchets, but also *backward secrecy* since compromised KDF keys will eventually be replaced by new seeds. To achieve *out-of-order resilience*, the Axolotl ratchet makes use of a second derivation function within its KDF ratchets. While the KDF ratchets are advanced normally, the KDF keys are passed through a second distinct derivation function before being used for encryption.

Figure 2.5 depicts the double ratchet used in Axolotl. The secondary KDF, denoted as KDF_2, allows the chain keys (c_i) to be advanced without sacrificing forward secrecy; each c_i is deleted immediately after being used to derive the subsequent chain key c_{i+1} and the corresponding message key (k_i) for encryption. If messages arrive out of order, this system provides a mechanism for decrypting the messages without compromising forward secrecy. For example, if Bob is expecting message M_1 and is storing c_1 in memory, but then receives M_2 instead, he uses c_1 to compute k_1, c_2, k_2, and c_3. Bob uses k_2 to decrypt the newly received message, and then he deletes c_1 and c_2 from memory, leaving only k_1 and c_3. When the missing M_1 eventually arrives, Bob can use k_1 to decrypt it directly. However, if an attacker compromises Bob's system at this moment, they will be unable to derive k_2 to decrypt M_2. A similar situation is depicted in Figure 2.5, where gray key nodes denote keys held in memory after Alice was able to receive M_4.

Axolotl also simplifies the use of its outer DH ratchet. In OTR, a chain of trust, allowing trust in new DH key exchanges to be traced back to the original AKE, is provided through the use of DH key advertisements and acknowledgments. To speed up this process, Axolotl instead derives a *root key* from the initial AKE in addition to the initial DH keys. Each subsequent DH secret is derived by using the sender's latest DH key, the latest DH key received from the other participant, and the current root key. Each time the DH ratchet is advanced, a new root key is derived in addition to a new chain key. Since deriving the chain keys requires knowledge of the current root key, newly received DH keys can be trusted immediately without first sending an acknowledgment. Despite these improvements, the double ratchet still requires *synchronicity* for the initial AKE.

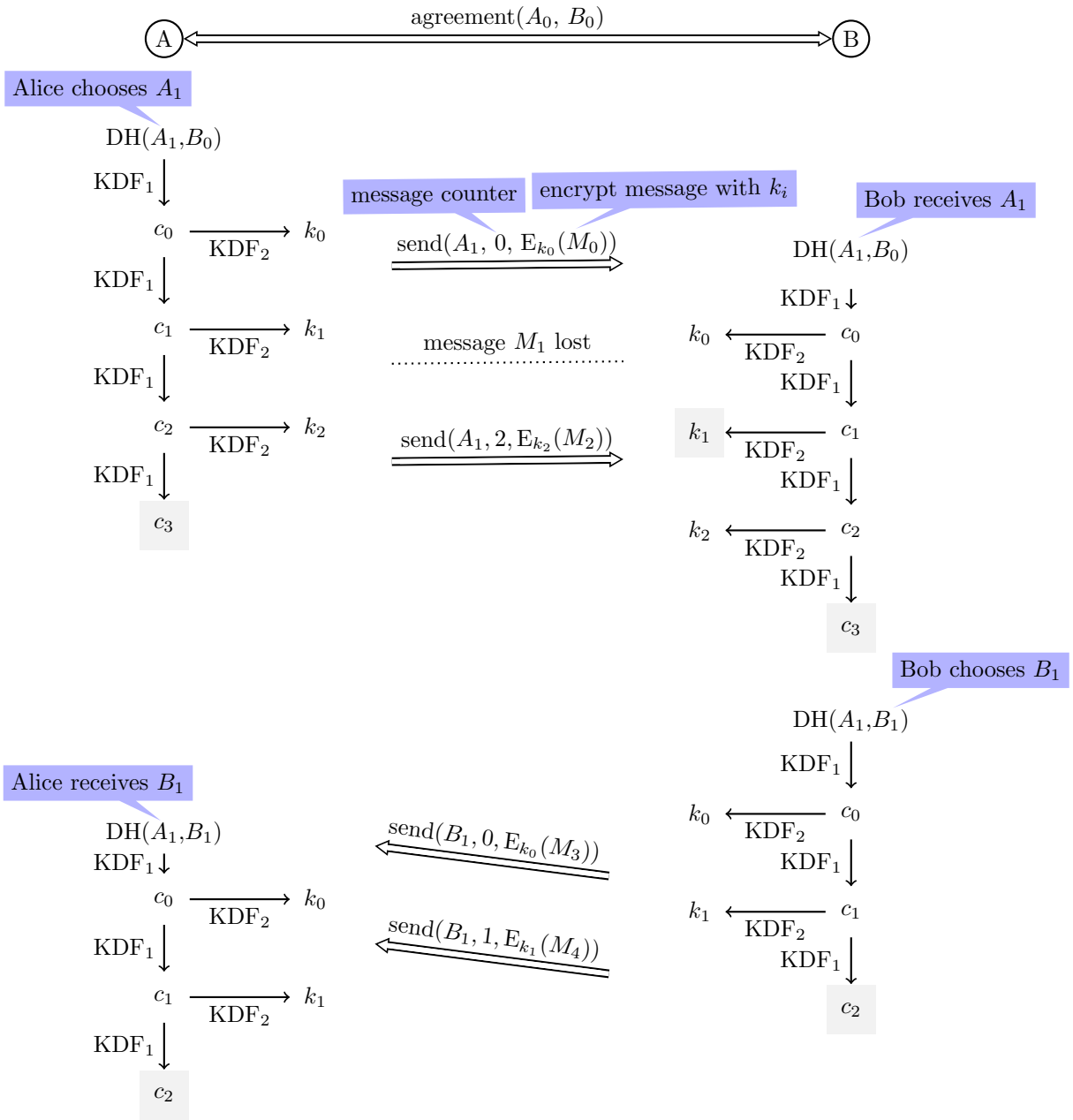


Figure 2.5: Simplified version of Axolotl: c_i denote chain keys, k_i message keys, KDF_i arbitrary key derivation functions, E_{k_i} an encryption function using k_i, and $A_i = g^{a_i}$ and $B_i = g^{b_i}$ as public DH values. Gray key nodes denote keys held in memory after Alice receives message M_4.

2.4.4.9 3-DH Handshake

A triple DH (3-DH) handshake is a different AKE scheme that provides stronger *participation repudiation* [Ope13d]. Specifically, transcripts of conversations between any two participants can be forged by anyone knowing nothing more than the long-term public keys of the participants. Figure 2.4b depicts a 3-DH AKE. Triple DH is an implicitly authenticated key agreement protocol—a category that has been extensively examined in the literature [MQV95; AJM95; Nat98; LMQ+03; Kra05; LM06; LLM07]. Assuming that Alice and Bob both have long-term DH keys g^a and g^b and ephemeral keys g^{a_e} and g^{b_e}, the 3-DH shared secret s is computed as $s = \text{KDF}(\text{DH}(g^{a_e}, g^{b_e})||\text{DH}(g^a, g^{b_e})||\text{DH}(g^{a_e}, g^b))$ [Per13]. If a secure key derivation function is used, a MitM attacker must either know a and a_e, or b and b_e. Kudla et al. [KP05] have shown that the 3-DH key exchange provides the same *authentication* level as achieved with the authenticated versions of DH key agreements. 3-DH achieves full *participation repudiation* since anybody is able to forge a transcript between any two parties by generating both a_e and b_e and performing DH key exchanges with g^a and g^b. Assuming that Mallory uses g^m as her long-term DH value and g^{m_e} as her ephemeral key agreement value, and that she knows Alice's long-term DH value g^a, she is able to forge a transcript by choosing g^{a_e} for Alice and calculating $s = \text{KDF}(\text{DH}(g^{a_e}, g^{m_e})||\text{DH}(g^a, g^{m_e})||\text{DH}(g^{a_e}, g^m))$ as the common HMAC and encryption secrets. Mallory can do this without ever actually interacting with Alice. Since the secret is partially derived from the long-term public keys, 3-DH also provides *participant consistency* without the need to explicitly exchange identities after a secure channel has been established. Unfortunately, this also causes a partial loss of *anonymity preservation* since long-term public keys are always observable during the initial key agreement (although future exchanges can be protected by using past secrets to encrypt these identities). It is possible to regain *anonymity preservation* by encrypting key identifiers with the given ephemeral keys.

2.4.4.10 Prekeys

While a *double ratchet* does not provide *asynchronicity* by itself, it can be combined with a *prekey* scheme to create an asynchronous version of the protocol. Prekeys are one-time ephemeral public DH contributions that have been uploaded in advance to a central server [Ope13b]. Clients can complete a DH key exchange with a message recipient by requesting their next prekey from the server. When combined with a 3-DH exchange, this is sufficient to complete an asynchronous AKE as part of the first message. In comparison to time-window based FS-IBE approaches (cf. Section 2.4.4.3), this approach requires the

precomputation of a number of ephemeral keys; otherwise, *forward secrecy* is weakened. However, this scheme also permits the destruction of the private ephemeral values immediately after receiving a message using them, instead of keeping a key until a time window expires.

TextSecure [Ope13c] is a popular Android app that combines Axolotl, prekeys, and 3-DH to provide an *asynchronous* user experience while sacrificing the *no additional service* property. It has gained considerable attention recently after being incorporated into WhatsApp [FMB+14; Ope14a]. Assuming Axolotl is used on two devices, the key material can evolve independently for each device. However, if one of those devices remains offline for a long time, a key compromise on that device is problematic: if the device can use its outdated keys to read messages that were sent when it was offline, then this compromise defeats *forward secrecy*; if the device cannot read the old messages, then the protocol does not achieve complete *multi-device support*. Deciding how long a device may be offline before it can no longer read buffered messages is an adoption consideration requiring further study of user behavior.

2.4.5 Group Chat Evaluation

2.4.5.1 Trusted Central Servers (Baseline)

The baseline protocol described in Section 2.4.4.1, where clients simply connect to a trusted central server using TLS, can trivially support group chats. While it is easy to add and remove group participants in this system, the only thing preventing participants from reading messages sent before or after they are part of the group is the trustworthiness of the server. This fact is indicated by half circles for *expandable / contractible membership*. SILC [SIL00] in its default mode is an example of a protocol using this design. While SILC's architecture involves a network of trusted servers similar to the IRC protocol, for analysis purposes this network can be considered as one trusted entity.

To improve the security and privacy of these systems, participants can simply encrypt and authenticate messages before sending them to the server by using a pre-shared secret key for the group. This approach is useful because it can be applied as a layer on top of any existing infrastructure. SILC has built-in support for this method in its "private mode"; users can provide a password for a channel that is used to derive a pre-shared key unknown to the server. While this design provides *confidentiality* and *integrity*, it does not provide *authentication*.

2.4.5.2 Key Transport

Rather than relying on users to exchange a secret password out-of-band, it is far better to automatically exchange a new secret for each conversation. A simple proposed method for doing this is to have one participant generate a session key and securely send it to the other participants. These systems begin by establishing secure channels between participants. The conversation initiator then generates a group key and sends it to the other participants using the pairwise channels. This design provides *forward* and *backward secrecy* since a new group key is randomly generated for each conversation. Due to the use of a group leader, *computational* and *trust equality* are also lost. However, groups are easily *expandable* and *contractible* by having the initiator generate and distribute a new group key.

An early design of this type, proposed by Kikuchi et al. [KTN04], suggests using a key directory to store static DH public keys for users. When group chats are formed, these keys and are used to derive pairwise session keys for the participants. A modified DH exchange is used in order to allow the server to reduce the required computation for the clients. *Participation repudiation* is lost due to the design of the key exchange mechanism, whose security properties have not been rigorously verified. An improvement, used in the GROK protocol [CKFP10], is to use standard DH exchanges for the pairwise channels, authenticated using long-term public keys stored in the key directory. This improvement provides *authentication* and *anonymity preservation*, but still suffers from the inherent inequality of key transport approaches.

2.4.5.3 Causality Preservation

One issue that is rarely addressed in the design of conversation security protocols is causality preservation. The user interface of the chat application must make design choices such as whether to display messages immediately when they are received, or to buffer them until causal predecessors have been received. However, the conversation security protocol must provide causality information in order to allow the interface to make these choices.

OldBlue [VC12] is a protocol that provides *speaker consistency* and *causality preservation*. An authenticated group key agreement (GKA) protocol is executed at the start of the conversation. Messages are encrypted with the group key and then signed with long-term asymmetric keys. This approach to signatures eliminates *message repudiation*. To preserve causality, messages include a list of identifiers of messages that causally precede them. The OldBlue protocol conservatively assumes that any message received by a user might influence the messages they subsequently send. Therefore, all received messages are considered to causally precede subsequently transmitted messages. Message identifiers are

hashes of the sender, the list of preceding identifiers, and the message contents. When a message has been lost, the client continuously issues resend requests to the other clients.

A different approach is employed by KleeQ [RKAG07], a protocol designed for use by multiple trusted participants with tenuous connectivity. An authenticated multi-party DH exchange is performed to initiate the protocol. By authenticating the parameters in a manner similar to OTR, *participation repudiation* can be provided. The group can be easily *expanded* by incorporating the DH contribution of a new member into the multi-party DH exchange, deriving a new group key. However, the group is not *contractible* without restarting the protocol. When two conversation participants can establish a connection, they exchange the messages that the other is missing using a patching algorithm. All messages are encrypted and authenticated with a MAC using keys derived from the group secret, providing *message repudiation*. Messages are sealed into blocks, which are sequences of messages having the property that no messages could possibly be missing. After each block is sealed, rekeying is performed using the previous keys and the block contents. A mechanism is provided to seal blocks even if some users are inactive in the conversation. *Speaker consistency* is not guaranteed until the messages have been sealed in a block. While participants are *authenticated* during group formation, message contents are not authenticated until after they have been sealed into a block. The block sealing mechanism indirectly provides *participant consistency* and *destination validation*. If malicious participants send differing messages to others, this will be uncovered during the block sealing phase. Manual resolution is required to identify malicious participants.

2.4.5.4 OTR Networks

Since OTR [BGB04] provides desirable features for two-party conversations, it is natural to extend it to a group setting by using OTR to secure individual links in a network. A basic strategy is to enlist a trusted entity to relay messages and then secure client links to this entity using OTR. This is the approach taken by the GOTR protocol released in 2007 (we write the year to distinguish it from a different protocol with the same name from 2013). GOTR (2007) [BST07] selects a participant to act as the relay, forming a star topology of pairwise connections with the selected participant acting as the hub. All *authentication* properties, *speaker consistency*, and *causality preservation* are lost because they do not persist across the relay node. Since the relay server can buffer messages, *asynchronicity* is provided as long as the relay node remains online. All other properties are inherited from OTR. Groups can be *expanded* and *contracted* simply by establishing new OTR connections to the relay.

Instead of using a star topology, pairwise OTR connections between all participants can be established. This approach restores *authentication* and *anonymity preservation*, as well as *equal trust* between members. It is also possible to send messages to *subgroups* by only transmitting the message across selected OTR links. The downside of this approach is that it does not *preserve causality* or provide *speaker consistency*; participants can send different messages to different people. This design also incurs significant computational overhead. It would be desirable to achieve these security properties without this level of additional cost.

2.4.5.5 OTR for Groups

Several protocols have been proposed to achieve OTR-like *repudiation* properties for group conversations. The TextSecure protocol can be naturally extended to groups by sending messages to each recipient using the two-party TextSecure protocol [Ope14b]. Multicast encryption is used for performance: a single encrypted message is sent to a central server for relaying to recipients while the decryption key for the message is sent pairwise using TextSecure. In practice, the wrapped decryption keys are attached to the same message for broadcasting. It is also possible to accomplish this task using one of the many existing broadcast encryption schemes [FN94]. This design does not provide any guarantees of *participant consistency*, but it inherits the *asynchronicity* of the two-party TextSecure protocol. *Speaker consistency* and *causality preservation* are achieved by attaching preceding message identifiers to messages. A message identifier is a hash of the sender, the list of preceding identifiers, and the message contents.

A *repudiable* group chat scheme can also be designed by utilizing a deniable group key exchange (DGKE) protocol, as in the mpOTR protocol [GUVC09; Van13]. When completed, the DGKE provides each participant with a shared secret group key and individual ephemeral signing keys. This information is authenticated with long-term keys in a manner providing *participation repudiation* while still *authenticating* participants—participants receive proof of each other's identities, but this proof cannot be used to convince outsiders. All parties must be online to complete the DGKE, so the protocol does not support *asynchronicity*. Messages are encrypted with the shared group key and signed with the ephemeral keys. The ephemeral signatures provide proof of authorship to others in the group but, because outsiders cannot be certain that these ephemeral signing keys correspond to specific long-term keys, *message repudiation* is preserved. However, since all messages from an individual are signed with the same (ephemeral) key, the protocol does not have *message unlinkability*. When the conversation has concluded, each participant hashes all messages received from each other participant. The hashes are then compared

to ensure that everyone received the same set of messages, providing *speaker consistency*. If this check fails, messages must be individually compared to uncover discrepancies. This approach, where a consistency check is performed only once at the conclusion of the conversation, does not work if a network adversary disconnects users from the conversation before the consistency check can be completed. In this worst-case scenario, the only information received by users is that something went wrong at some point during the protocol, but nothing more specific. Unfortunately, in many scenarios it is unclear how users should respond to this limited information. In this scheme, *subgroup messaging* is not possible since all messages share a single encryption key. The group is also not *expandable* or *contractible* without performing a new DGKE.

A completely different approach is taken by the GOTR (2013) protocol. GOTR (2013) [LVH13] is built using a "hot-pluggable" group key agreement (GKA) protocol, allowing members to join and drop out of the conversation with little overhead. This system involves the use of "circle keys": sets of public keys having the property that a shared secret key can be computed by anyone with a private key matching a public key in the set. The key exchange mechanism in this protocol is relatively complex; we refer the interested reader to the original publication for details [LVH13]. Pairwise secure channels are set up between participants to send consistency check messages. These consistency channels have the effect of providing *global transcript order*, but all participants are required to be online to receive messages. The system otherwise provides features similar to mpOTR but with *flexible group membership* and *message unlinkability*.

2.4.6 Discussion

Similar to our study of trust establishment, Table 2.2 makes immediately clear that no conversation security protocol provides all desired properties. Since most of the properties in the table are not mutually exclusive, however, there is significant room for improvement by combining protocol designs and this should be seen as a tangible and important call to action for the research community.

Sadly, the most widely adopted solutions also have the worst security and privacy properties, with most non-security-focused applications providing only basic static asymmetric cryptography. This does not appear to be due to the usability drawbacks of the more secure protocols: once the trust establishment has been done, all of the conversation security approaches we studied can be automated without any additional effort for the user. An exception is enabling *asynchronous* communication while still providing *forward* and *backward secrecy*; the only solution for this problem that appears to have any significant

deployment in practice is the prekeys approach implemented by TextSecure. This requires relatively complicated infrastructure compared to a simple key server, introduces problems for *multi-device support*, and is prone to denial-of-service attacks if it is used in anonymous communication. This approach is poorly studied in the academic literature. The FS-IBE scheme discussed in Section 2.4.4.3 promises to resolve the issues of server complexity and denial of service, but introduces new challenges such as scalability and performance issues [CHK03]. Unlike prekeys (Section 2.4.4.10), this scheme has received a considerable amount of follow-up research and academic citations, but we are unaware of any practical tool implementing it. In addition, a time-window based FS-IBE scheme requires holding the ephemeral keys for a certain amount of time to allow decryption of delayed messages. One possible mitigation is to rely on an additional server maintaining window counters where every window number is used once, analogous to the prekeys approach. Improving the practicality of FS-IBE and puncturable encryption schemes warrants further research.

Another outstanding concern that limits adoption of secure conversation security protocols is the limited support for multiple devices. Despite a vast number of users owning multiple devices, only the most insecure protocols support this property without requiring users to perform pairing procedures. Device pairing has proved extremely difficult for users in practice [KFR09; War14] and allowing users to register multiple devices with distinct keys is a major usability improvement. Although extremely difficult, implementing usable device pairing is not necessarily an insurmountable problem. Additional work in this area is needed.

When it comes to group chat properties, we can identify several areas for improvement in Table 2.2. Classic protocols often do not provide *participant consistency* or *destination validation*, making them potentially vulnerable to surreptitious forwarding or identity misbinding attacks. However, these are sometimes addressed in concrete implementations. The double ratchet used in Axolotl improves *forward secrecy* with low cost in performance, implementation complexity, and resilience, but it has not yet been thoroughly evaluated in an academic context. Additionally, decentralized group chat systems inherently permit a participant to send different messages to different people. Due to network conditions, users can also end up observing significantly different transcripts. Despite these intrinsic weaknesses, surprisingly few protocols explicitly consider *speaker consistency* or *causality preservation*. The recently proposed (n+1)sec protocol [eQu15] is an example of new work in this area. (n+1)sec builds off of the flexible group key exchange protocol of Abdalla et al. [ACMP10] to provide a DGKE and checks for transcript consistency.

Existing solutions achieve mixed results concerning *repudiation*. For the definitions of *participation repudiation* and *message repudiation* used in this chapter, the two-party protocols based on authenticated DH key exchanges and the OTR-like group protocols

provide inexpensive solutions. However, there exist stronger definitions of deniability that none of the evaluated approaches adequately satisfies. We return to this issue in Chapter 3.

There are also additional adoption constraints imposed by many modern secure group chat protocols. Group protocols often choose to employ either a trusted participant or an additional service to improve protocol performance, which can lead to security concerns or introduce additional costs for deployment. Very few group protocols support *subgroup messaging* or changing group membership after the conversation has started without incurring the substantial costs of a new protocol run. Additionally, many proposed designs require synchronicity in order to simplify their protocols, which largely precludes their use on current mobile devices.

2.5 Transport Privacy

The transport privacy layer defines how messages are exchanged, with the goal of hiding message metadata such as the sender, receiver, and conversation to which the message belongs. Some transport privacy architectures impose topological structures on the conversation security layer, while others merely add privacy to data links between entities. The transport privacy schemes may also be used for privacy-preserving contact discovery. In this section, we compare approaches for transport privacy in terms of the privacy features that they provide, as well as usability concerns and other factors that limit their adoption. Table 2.3 compares the various schemes.

2.5.1 Privacy Features

We make the distinction between *chat messages*, which are the user-generated payloads for the messaging protocol to exchange, and *protocol messages*, which are the underlying data transmissions dictated by the upper protocol layers. We define the following privacy properties:

- **Sender Anonymity**: When a chat message is received, no non-global entities except for the sender can determine which entity produced the message.
- **Recipient Anonymity**: No non-global entities except the receiver of a chat message know which entity received it.
- **Participation Anonymity**: No non-global entities except the conversation participants can discover which set of network nodes are engaged in a conversation.

Scheme	Example	Privacy							Usability								Adoption					
		Recipient Anonymity	Sender Anonymity	Particip. Anonymity	Global Adv. Unlinkability	Resistant	No Message Contact Discovery	No Message Delays	Easy Initialization	No Message Drops	No Fees Required	Topology Independent	No Additional Service	Spam/Flood Resistant	Low Storage	Low Bandwidth	Low Computation	Asynchronous	Scalable			
Store-and-Forward†*	Email/XMPP	-	-	-	-	-	●	◐	●	●	●	●	-	-	●	●	●	●	●			
+DHT Lookup†*	Kademlia	◐	◐	-	-	-	●	◐	●	●	●	●	●	◐	●	◐	●	●	●			
Onion Routing+Message Padding†*	Tor	●	-	●	●	-	-	◐	●	●	●	●	◐	-	●	●	●	-	●			
+Hidden Services*	Ricochet	●	●	●	◐	-	-	◐	●	●	●	●	◐	-	●	●	●	-	●			
+Inbox Servers†	-	●	-	●	●	-	-	◐	●	●	●	●	-	-	●	●	●	●	●			
+Random Delays†*	Mixminion	●	-	●	●	◐	-	-	●	●	●	●	-	-	◐	●	●	●	●			
+Hidden Services+Delays+Inboxes+ZKGP*	Pond	●	-	●	●	◐	-	-	●	●	●	●	-	●	◐	●	●	●	●			
DC-Nets†*	-	●	●	-	-	●	-	-	●	●	●	●	-	●	-	●	●	●	-	-		
+Silent Rounds†	Anonycaster	●	●	-	-	●	-	-	●	●	●	●	-	●	◐	●	●	●	-	-		
+Shuffle-Based DC-Net+Leader†	Dissent	●	●	-	-	●	-	-	●	●	●	●	-	●	●	●	●	●	-	-		
+Shuffle-Based DC-Net+Anytrust Servers†	Verdict	●	●	-	-	●	-	-	●	●	●	●	-	-	●	●	●	●	-	◐		
Message Broadcast†	-	-	●	●	●	●	●	●	●	●	●	●	●	●	-	-	-	◐	-	-		
+Blockchain	-	◐	●	●	●	●	●	-	-	●	-	●	◐	●	-	-	-	-	●	-		
PIR*	Pynchon Gate	-	●	●	●	●	●	-	●	◐	●	●	-	-	-	◐	◐	●	●	◐		

● = provides property; ◐ = partially provides property; - = does not provide property;
†has academic publication; *end-user tool available

Table 2.3: Transport privacy schemes. Every privacy-enhancing approach carries usability and/or adoption costs.

- **Unlinkability**: No non-global entities except the conversation participants can discover that two protocol messages belong to the same conversation.
- **Global Adversary Resistant**: Global adversaries cannot break the anonymity of the protocol.

2.5.2 Usability Properties

- **Contact Discovery**: The system provides a mechanism for discovering contact information.
- **No Message Delays**: No long message delays are incurred.
- **No Message Drops**: Dropped messages are retransmitted.
- **Easy Initialization**: The user does not need to perform any significant tasks before starting to communicate.
- **No Fees Required**: The scheme does not require monetary fees to be used.

2.5.3 Adoption Properties

- **Topology Independent**: No network topology is imposed on the conversation security or trust establishment schemes.
- **No Additional Service**: The architecture does not depend on availability of any infrastructure beyond the chat participants.
- **Spam/Flood Resistant**: The availability of the system is resistant to denial-of-service attacks and bulk messaging.
- **Low Storage Consumption**: The system does not require a large amount of storage capacity for any entity.
- **Low Bandwidth**: The system does not require a large amount of bandwidth usage for any entity.
- **Low Computation**: The system does not require a large amount of processing power for any entity.

- **Asynchronous**: Messages sent to recipients who are offline will be delivered when the recipient reconnects, even if the sender has since disconnected.
- **Scalable**: The amount of resources required to maintain system availability scales sublinearly with the number of users.

2.5.4 Evaluation

2.5.4.1 Store-and-Forward (Baseline)

To evaluate the effectiveness and costs of different transport privacy architectures in Table 2.3, we compare the solutions to a baseline. For the baseline protocol, we assume a simple store-and-forward messaging protocol. This method is employed by email and text messaging, causing minor message delays and storage requirements for intermediate servers. Since email headers contain sender and recipient information, a simple store-and-forward mechanism does not provide any privacy properties.

2.5.4.2 Peer-to-Peer Solutions

Instead of relying on centralized servers for message storage and forwarding, peer-to-peer based schemes try to establish a direct message exchange between the participants. Since end users frequently change their IP addresses, these systems often use Distributed Hash Tables (DHTs) to map usernames to IP addresses without a central authority. Examples of popular DHT systems are Chord, Kademlia (used by BitTorrent), and GNUnet [SML+01; MM02; PGES05]. In addition to acting as an IP address lookup table, it is possible to store exchanged messages directly in a DHT. Various query privacy extensions have been proposed to prevent other users from learning what data is being requested. They can be used in advanced DHT overlays allowing anonymous queries and message exchange [KT08; WB12; BGKT12].

Global network adversaries are still able to see the traffic flow between participants during message exchange. Thus, clients have two options to protect the data flow: fake message transmissions, or use anonymization techniques. End-user clients might use services such as onion routing, which is evaluated in the next section, to hide their identities.

From the usability and adoption perspective, peer-to-peer networks require a synchronous environment. DHTs can be used for *contact discovery* with *easy initialization*, but they introduce *message delays* and *message drops*.

In practice, various end-user applications use the BitTorrent or GNUnet networks for their secure messaging service. For instance, Tox, Bleep, and other messengers use BitTorrent for message exchange. The GNUnet Name Service (GNS) offers privacy-preserving name queries for contact discovery [WSG14a].

2.5.4.3 Onion Routing

Onion routing is a method for communicating through multiple proxy servers that complicates end-to-end message tracing [RSG98]. In onion routing, senders send messages wrapped in multiple layers of encryption through preselected paths—called *circuits*—of proxy servers. These servers unwrap layers of encryption until the original message is exposed, at which point it is relayed to the final destination. Each node in the path only knows the immediate predecessor and successor in the path. The routing process adds some latency to messages, but otherwise retains the baseline usability features. An onion routing protocol, such as the widely used Tor protocol [DMS04], provides *sender anonymity*, *participant anonymity*, and *unlinkability* against network attackers with limited scope.

Global network adversaries are still able to break the anonymity properties of simple onion routing designs by performing statistical analysis incorporating features such as content size, transmission directions, counts, and timing, among others. The success of such an adversary can be limited by individually eliminating these features. Protection can be added, for example, by introducing random delays to transmissions. The longer the allowed delays, the less statistical power is available to the adversary. Of course, this imposes potentially long *message delays* and *additional storage requirements* for relays, making it unusable for synchronous instant messaging.

Unfortunately, random delays do not completely defeat global adversaries. The only way to do so is to make transmission indistinguishable from no transmission (e.g., by saturating the bandwidth of all connections). However, in practice, this is likely infeasible. Additionally, concrete implementations such as Tor often provide weaker anonymity guarantees than idealized onion routing schemes. Several prominent attacks against Tor have been based on implementation defects, limited resources, weaknesses introduced by performance trade-offs, and predictability of the content being transmitted [MD05; BMG+07; EDG09; PNZE11]. Adoption of onion routing is limited by the requirement to establish a large network of nodes to provide a sufficient anonymity set and cover traffic.

In the default mode, onion routing systems do not provide *recipient anonymity*. However, Tor can be modified to achieve this property using an extension called *hidden services*. To create a Tor hidden service, the recipient uses traditional Tor circuits to upload a set of

introduction points and a public key to a public database. The sender later uses a circuit to acquire this information from the database. The sender chooses a *rendezvous point* and sends it along with a nonce to the recipient through an introduction point. The recipient and sender both connect to the rendezvous point, which uses the nonce to establish a communication channel by matching up the sender and recipient circuits.

To provide *asynchronous* communication support, store-and-forward servers can be incorporated into the onion routing model. Each user is associated with a Tor hidden service that remains online. To send a message, the sender constructs a circuit to the recipient's server and transmits the message. Users periodically poll their own servers to determine if any messages are queued. Ricochet is an example of this approach [Ric14].

Pond uses this design for its transmission architecture [Lan13b] but adds random delays between connections, all of which transmit the same amount of data, to weaken statistical analysis by network adversaries. While some protection against global network adversaries is provided by the onion routing model, this protection is strictly weaker than Tor because connections are made directly from senders to recipient mail servers. This design requires *storage commitments* by servers and also introduces very *high latency*.

Without additional protections, this scheme is also highly vulnerable to denial-of-service attacks because connection delays and fixed transmission sizes artificially limit bandwidth to very low levels. Pond addresses this by requiring users to maintain group lists secured by zero-knowledge-group-proof schemes (ZKGP). This way, recipients can upload contact lists without revealing their contacts. Simultaneously, senders can authenticate by providing zero-knowledge proofs that they are in this list. The BBS signature scheme [BBS04] is currently used by Pond to achieve this. Additional work is underway to provide a similar mechanism in more efficient manner by using one-time delivery tokens [Lan13b].

The ZKGP schemes used by Pond are related to secret handshake protocols. Secret handshakes enable authentication between parties that share some attributes, while keeping identities hidden from others [BDS+03].

2.5.4.4 DC-nets

Dining Cryptographer networks (DC-nets) are anonymity systems that are often compared to onion routing schemes. Since they are primarily used as a general-purpose transport privacy mechanism, many varieties have been proposed [Cha88; WP89; GRPS03; GJ04; CF10; Hea12; WCFJ12; CWF12; SCW+14]. We focus on recently introduced schemes that explicitly list secure messaging as an intended use case.

DC-nets are group protocols that execute in rounds. At the start of each round, each participant either submits a secret message or no message. At the end of the round, all participants receive the xor of all secret messages submitted, without knowing which message was submitted by which participant. In this way, DC-nets provide *sender anonymity* while also achieving *global adversary resilience*—no statistical analysis can reveal the sender of a message. *Recipient anonymity* can be achieved by using the protocol to publish an ephemeral public key. Messages encrypted with this key are then sent and, since the owner of the matching private key is unknown, the participant able to decrypt the messages cannot be determined. Since messages are sent in rounds, DC-nets add message latency and do not support *asynchronous* communication; dropped messages prevent the protocol from advancing. Messages are easily *linked* by observing which network nodes participate in a round. Additionally, DC-nets have limited *scalability* due to requiring pairwise communication.

The basic DC-net design has a problem with collisions: if two parties submit a message in the same round, the result will be corrupted. A malicious participant can exploit this to perform an anonymous denial-of-service attack by submitting garbled messages each round. Worse still, an active network attacker can also perform this attack by perturbing transmitted bits. There are several approaches to mitigate this problem. Anonycaster [Hea12] adds pseudorandomly determined "silent rounds" where all members know that no message should be contributed. Receipt of a message during a silent round indicates a denial-of-service attack by an active network attacker. However, malicious participants can still launch attacks by sending garbled messages only during non-silent rounds.

Dissent [CF10; WCFJ12; SCW+14] and Verdict [CWF12] take a different approach by constructing a DC-net system through the use of a verifiable shuffle and bulk transfer protocol. Shuffle-based DC-nets can include a blame protocol to pinpoint the entity that caused a round to fail. Dissent appoints one participant as a leader to manage round timing, the blame protocol, and exclusion of disconnected members from rounds, thereby restoring support for *asynchronicity*. Verdict uses an alternative approach where the DC-net protocol is executed by a set of central servers that clients connect to, providing greater *scalability* and maintaining security as long as any one server is honest.

While DC-nets are primarily a transport privacy mechanism, they are distinguished from other schemes by their use of rounds and the fact that every network node is also a participant in the conversation. When using DC-nets to transmit higher-level conversation security protocols, it is important for designers to consider how these properties affect the overall security of the scheme (e.g., the use of synchronous rounds creates a *global transcript*, and the details of the DC-net key exchanges may cause a loss of *participation repudiation*).

2.5.4.5 Broadcast Systems

There is a simple approach to providing recipient anonymity against all attackers, including global adversaries: distributing messages to everyone. This approach provides *recipient anonymity*, *participation anonymity*, and *unlinkability* against all network attackers. It also provides a natural way to *discover contacts* because requests for contact data can be sent to the correct entity without knowledge of any addressing information. However, there are some serious downsides that hinder adoption: broadcasting a message to everyone in the network requires high bandwidth, there is no support for *asynchronicity*, and it has extreme *scalability* issues. Additionally, it is easy to attack the availability of the network through flooding. Bitmessage [Bit12], a broadcast-based transport system, either *requires monetary fees* or a proof of work to send messages in order to limit spam, adding *computation requirements* and *message delays* as represented by the *blockchains* row in Table 2.3. It is also possible to alleviate *scalability* problems by clustering users into smaller broadcast groups, at the cost of reduced anonymity set sizes.

2.5.4.6 PIR

Private Information Retrieval (PIR) protocols allow a user to query a database on a server without enabling the server to determine what information was retrieved. These systems, such as the Pynchon Gate [SCM05], can be used to store databases of message inboxes, as well as databases of contact information. *Recipient anonymity* is provided because, while the server knows the network node that is connecting to it, the server cannot associate incoming connections with protocol messages that they retrieve. For the same reason, the protocols offer *participation anonymity* and *unlinkability*. By default, there is no mechanism for providing *sender anonymity*. These systems are naturally *asynchronous*, but they result in *high latency* because inboxes must be polled. The servers also incur a *high storage cost* and are vulnerable to flooding attacks.

PIR schemes can also be used to privately retrieve presence information, which can be useful for augmenting synchronous protocols lacking this capability. For example, DP5 [BDG14] uses PIR to privately provide presence data for a secure messaging protocol; DP5 does not facilitate message transmission itself.

PIR implementations can be divided into computational schemes, which rely on computational limitations of the server, information-theoretic schemes, which rely on non-collusion of servers, and hybrid schemes that combine properties of both. There is also a class of PIR schemes that makes use of secure coprocessors, which requires users to trust that the (remote) coprocessor has not been compromised. PIR implementations differ in

their bandwidth, computation, and initialization costs, as well as their scalability. PIR is not widely adopted in practice because one or more of these costs is usually prohibitively expensive.

2.5.5 Discussion

If messages are secured end to end, leaving only identifiers for anonymous inboxes in the unencrypted header, then metadata is easily hidden from service operators. Assuming that each message is sent using new channels, an adversary is not able to link single messages to conversations. However, such schemes introduce adoption and usability issues; they are prone to spam, flooding, and denial-of-service attacks, or require expensive operations such as zero-knowledge authentication, posing barriers to adoption. Worse still, hiding metadata from a global adversary in these schemes necessitates serious usability problems such as long delays.

In contrast, decentralized schemes either exhibit synchronicity issues or have serious scalability problems. Most decentralized projects, especially BitTorrent-based approaches, lack detailed documentation that is required for complete evaluation. Some tools claiming to hide metadata only do so in the absence of global network adversaries, which recent surveillance revelations suggest may exist.

Broadcast-based schemes can achieve the best privacy properties, but exhibit serious usability issues, such as lost or delayed messages, in addition to apparently intractable scalability issues. Even if anonymous transmission schemes are adopted, they require a large user base to provide a high degree of anonymity, potentially discouraging early adopters. Finally, care must be taken when selecting a conversation security scheme to avoid leaking cryptographic material or identifiers that might lead to deanonymization.

2.6 Future Directions

The vast majority of the world's electronic communication still runs over legacy protocols such as SMTP, SMS/GSM, and centralized messengers, none of which were designed with end-to-end security in mind. We encourage the research community to view the high-profile NSA revelations in the United States as a golden opportunity to encourage the adoption of secure systems in their place. As the old adage goes: "never let a crisis go to waste".

Unfortunately, while we have seen considerable progress in practical tools over the past two years, there is little evidence suggesting that academic research on secure messaging

has dramatically increased. This is unfortunate for two reasons: first, many interesting problems of practical importance remain unresolved. In particular, apparent practical deployment constraints, including limitations for asynchronous communication, multiple independent devices, and zero user effort, are not fully appreciated in most published research papers. Second, many theoretically *solved* problems are not considered in practice, whether because developers are unaware of their existence, or because they cannot immediately translate the cryptographic publications into working systems.

Our effort to systematize existing knowledge on secure messaging suggests three major problems must be resolved: *trust establishment*, *conversation security*, and *transport privacy*. The schemes can largely be chosen independently, yielding a vast design space for secure messaging systems. Yet we also caution against a proliferation of à-la-carte systems for specific niches. The main purpose of communication networks is to interact with others and there is considerable value in having a small number of popular protocols that connect a large number of users. Currently, many people fall back to email despite its insecurity.

We also note that, disappointingly, most of the exciting progress being made right now is by protocols that are either completely proprietary (e.g., Apple iMessage) or are open-source but lack a rigorously specified protocol to facilitate interoperable implementations (e.g., TextSecure). An open standard for secure messaging, combining the most promising features identified by our survey, would be of immense value.

Inevitably, trade-offs have to be made. We conclude that secure approaches in trust establishment perform poorly in usability and adoption, while more usable approaches lack strong security guarantees. We consider the most promising approach for trust establishment to be a combination of central key directories, transparency logs to ensure global consistency of the key directory's entries, and a variety of options for security-conscious users to verify keys out of band to put pressure on the key directory to remain honest.

Our observations on the conversation security layer suggest that asynchronous environments and limited multi-device support are not fully resolved. For two-party conversation security, per-message ratcheting with resilience for out-of-order messages combined with deniable key exchange protocols, as implemented in Axolotl, can be employed today at the cost of additional implementation complexity with no significant impact on user experience. The situation is less clear for secure group conversations; while no approach is a clear answer, the TextSecure group protocol provides pragmatic security considerations while remaining practical. It may be possible to achieve other desirable properties, such as participant consistency and anonymity preservation, by incorporating techniques from the other systems. It remains unclear exactly what consistency properties are required to match users' expectations and usability research is sorely needed to guide future protocol

design. Finally, transport privacy remains a challenging problem. No suggested approaches managed to provide strong transport privacy properties against global adversaries while also remaining practical.

The active development of secure messaging tools offers a huge potential to provide real-world benefits to millions. Our goal in this chapter was to support this development by systematizing secure messaging research and highlighting several areas for future work. Next, we will focus our attention on one of these open problems: improving the deniability properties of existing conversation security schemes.

Chapter 3

Deniability for Secure Messaging

In the original publication of Off-the-Record Messaging, Borisov et al. argued that unrestricted non-repudiation is an undesirable property for secure messaging protocols [BGB04]. Instead, a better approach is to provide authentication only for the parties taking part in the protocol, while preventing the transfer of proofs of authorship to other parties. If observers of the protocol cannot be certain that the conversation was genuine, then the conversation participants can speak without fear of their statements being plausibly attributed to them by third parties. In other words, the participants can speak "off-the-record". If a protocol provides this property, then we say that it offers "deniability" or "repudiation".

Since the release of OTR, many secure messaging protocols have attempted to provide deniability. However, different schemes define deniability in slightly different ways. Additionally, no existing secure messaging schemes can be said to be deniable under all definitions. This incomprehensiveness is despite the fact that these definitions are largely orthogonal, and deniability properties can be implemented without any effects on the usability of the protocol (since they are provided by the conversation security component).

In this chapter, we discuss various definitions of deniability that have been proposed in the literature. We then turn our attention to deniable authenticated key exchanges—a class of cryptographic protocols that can be used to construct deniable secure messaging schemes. There exists only one known key exchange protocol that satisfies all of our deniability definitions, but it has never been implemented. We discuss this existing protocol and present an improvement that makes its use practical in many environments. We then construct two new efficient deniable key exchange protocols designed to provide strong repudiation in modern messaging environments (e.g., asynchronous smartphone communications). We focus solely on two-party protocols in this chapter.

3.1 Deniability

Deniability is a notoriously difficult concept to define. This problem arises due to the fact that deniability is actually a series of distinct, but related, properties. When we discuss deniability, we must do so with respect to an action and a type of judge.[1] We say that an action is deniable with respect to a given judge if the judge cannot be convinced that an individual performed the action. To make such a statement, we need to define the environment in which the judge resides, and the type of evidence that is required to convince the judge that the action was performed. If an action is deniable with respect to a judge, we can say that individuals can "plausibly deny" performing the action (with the definition of "plausibility" being determined by the requirements of the given judge). In this chapter, we focus on interesting actions and judges within the context of secure messaging protocols.

3.1.1 Deniable Conversations

There are two primary aspects of conversations that can be called deniable. We can say that any messages transmitted in a conversation are deniable (*message repudiation*), but we can also say that participation in the conversation itself is deniable (*participation repudiation*). A secure messaging protocol that offers *message repudiation* but not *participation repudiation* with respect to a given judge allows the judge to conclude that two users communicated with each other, but not that a given message was exchanged as part of that conversation. When a user participates in a protocol offering *participation repudiation* but not *message repudiation*, the situation is more nuanced. The user can plausibly deny taking part in the conversation, but if a judge somehow becomes convinced that they *did* participate, then they cannot deny the transmission of the messages within the conversation. For example, if Alice benefits from admitting to the judge that she spoke to Bob on a given date, but she would like to deny sending a particular message as part of that conversation, she would not be able to so in a protocol lacking *message repudiation*.

[1] When we refer to judges, we are referring to entities that decide whether or not a certain event occurred. We are not referring to judges in the sense of the legal profession; however, judges of the latter type are often judges of the former type.

3.1.2 Judges

We now turn to the task of defining judges relevant to deniability in secure messaging protocols.

3.1.2.1 Plausible Deniability

When defining a judge, we must address the issue of plausibility. Namely, we must define the conditions under which the judge will believe that a user performed a given action, such as sending a message or participating in a conversation (and correspondingly, the situations in which performing the action is plausibly deniable).

Unfortunately, we must make some assumptions about the behavior of judges if we wish to make meaningful statements about the deniability of secure messaging protocols. If we allow judges to use any criteria to deliver judgments, then we can never conclude that an action is deniable (e.g., we cannot enable a user to plausibly deny performing an action to a judge that arrives at conclusions randomly). However, we must ensure that we model the behavior of judges in a realistic way. If we require a judge to have an unrealistically large amount of evidence to be convinced of an action, then we may admit protocols that are not deniable in practice. In contrast, underestimating the amount of evidence needed by a judge may unrealistically exclude protocols from being called deniable.

In the secure messaging literature, it is common to consider only judges that behave in one specific manner. These judges are completely rational, and decide on the plausibility of an event based solely on the evidence presented to them. Moreover, the only acceptable evidence for these judges is a valid cryptographic proof showing that the event must have occurred. Normally, this involves an unforgeable cryptographic signature, generated using secret keys known to be available only to the user in question, that testifies to the action in question, and that is verifiable by the judge.

Note that unencrypted messages sent between Internet users (e.g., unencrypted IRC conversations) provide both message and participation repudiation against a judge of this type. This is intuitively true because anyone can forge a transcript of such a communication, and thus it would not constitute convincing evidence. It is also important to note that the model of the judge should take practical considerations into account when assessing the implications of evidential cryptographic proofs. For example, if the judge receives cryptographic proof that a message was either sent by Alice or by Mallory, under the condition that Mallory had access to unrealistically powerful computational resources, then Alice may not be able to convincingly deny sending the message in practice. In contrast, a proof

showing that a message was either sent by Alice or by one other unspecified individual with commonly available resources is likely to provide far more plausible deniability in practice.

In this chapter, we consider only protocols that offer very strong deniability. These protocols allow conversation transcripts to be forged by any user with the ability to perform basic computations, while still providing authentication to conversation participants. Consequently, no unforgeable cryptographic proofs can be produced to convince a judge that sessions of these protocols took place.

3.1.2.2 Judge Positioning

In addition to the requirements for evidence, we must also define the circumstances of the judge. Specifically, we must define their relationship to the participants in the secure messaging protocol, and their capabilities. There are two primary types of judges that have been discussed in the secure messaging literature: offline judges, and online judges.

An **offline judge** examines the transcript of a protocol execution that occurred in the past, and decides whether or not the event in question occurred. A judge of this type is given a *protocol transcript*, showing all of the (usually encrypted) data transmitted between participants, and a *chat transcript*, showing the high-level chat messages that were exchanged. The judge must then decide whether the protocol and chat transcript constitute proof that the action in question occurred (e.g., a given user sent a given message, or two given users communicated with each other using the secure messaging protocol). When proving the deniability of protocols, it is also normally assumed that an offline judge is given access to the long-term secrets of all parties named in the transcript; judges should not be able to distinguish real transcripts from fake ones even when given access to these secret keys. Since a judge with access to these long-term secrets has at least as much distinguishing power as the same judge without this access, designing protocols that achieve this level of deniability ensures that judges have no incentive to compromise long-term secrets in practice. Chapter 2 exclusively considered judges of this type.

Typically, deniability with respect to offline judges is provided by allowing others to produce forged chat and protocol transcripts. If a protocol transcript can be forged, then the protocol provides participation repudiation. If an alternate chat transcript can be forged for a given protocol transcript, then the protocol provides message repudiation. Security proofs for these properties typically demonstrate that a simulator can produce transcripts that a judge cannot distinguish from real transcripts.

An **online judge** interacts with a protocol participant, referred to as the *informant*, while the secure messaging protocol is being executed. The judge has a secure and private

connection to the informant, and may instruct the informant to perform actions in the protocol. The goal of the judge is to evaluate whether the actions of other participants in the protocol are actually occurring, or if the informant is fabricating the conversation (i.e., they are actually a *misinformant*). The judge does not have any direct visibility into the network, but it may instruct the informant to corrupt participants. The judge is also informed whenever a participant has been corrupted. This situation can be likened to a real-world situation in which the informant is "wearing a wire" and an earpiece providing secure communication to a judge in another physical location.

3.1.3 Practicality

There has previously been some debate within the secure messaging community as to whether or not deniability should be implemented in end-user tools [Hea14]. There are two main arguments against designing deniable messaging protocols: deniability properties are too expensive to implement, given their benefits, and these properties are not useful in practice. As we explained in Chapter 2, the relevant deniability properties for secure messaging protocols are part of the conversation security component, and thus carry no usability consequences. We argue that incorporating some deniability properties into secure messaging proocols is now reasonably inexpensive due to the availability of efficient deniable key exchange protocols offering offline repudiation (see Chapter 2).

While the secure messaging literature mostly focuses on judges that understand cryptography and rely on cryptographic proofs to make decisions, real-world judges often do not behave in this manner, and routinely accept plaintext transcripts. We cannot design protocols that provide more deniability than plaintext; however, we can easily design protocols that provide *less* deniability—while a real-world judge of this type may not accept arguments that a plaintext transcript could theoretically be forged, they may be likely to accept testimony from experts that a protocol containing a digital signature could *not* be forged.

3.2 Deniable Authenticated Key Exchanges

Most secure messaging solutions incorporate an authenticated key exchange (AKE) protocol as part of their construction. At a high level, the goal of an AKE is to establish a fresh shared session key, and to authenticate the conversation participants to each other. When designing an AKE, we assume that trust establishment has already been performed

and all users of the system have well-known long-term public keys associated with them. If the session key cannot be derived from a protocol transcript even when the long-term secret keys are compromised in the future, then the AKE is said to have *forward secrecy*. A deniable authenticated key exchange (DAKE) is an AKE that additionally allows participants to plausibly deny taking part in the protocol. DAKEs have been widely studied in the literature, and we briefly survey some of the existing protocols in this section.

Bellare and Rogaway first formalized the definition of AKEs two decades ago [BR93]. While many AKE schemes were published before and after this definition, deniability for key exchanges was not a major concern. In the years that followed, deniability emerged as a desirable property for authentication protocols. Dolev et al. presented the first explicitly deniable authentication protocol [DDN98]. Deniability was later formalized as a property for authentication protocols by Dwork et al. [DNS98]. Since then, numerous papers have expanded this work; see, for example, Dwork and Naor [DN00b], Naor [Nao02], Katz [Kat03], Pass [Pas03], Wang and Song [WS09], and Youn, Lee, and Park [YLP11].

Shortly after the formalization of the AKE concept, several protocols claimed to offer deniability informally [Kra96; Kra03; BMP04]. Each of these protocols lacks some aspect of deniability covered in Section 3.1. The SKEME protocol [Kra96] lacks deniability against online judges; an online judge can insist on generating ephemeral keys for the informant, thereby preventing simulators from learning the shared secret. The SIGMA protocols [Kra03] lack online repudiation because they involve non-repudiable signatures that cannot be simulated by a misinformant. While the scheme constructed by Boyd et al. [BMP04] provides online repudiation, protocol transcripts can only be forged offline by alleged participants in the protocol (thus, its deniability against offline judges is limited).

With the release of the Off-the-Record Messaging protocol in 2004, repudiation was recognized as a desirable feature for secure messaging applications [BGB04]. Shortly thereafter, Di Raimondo et al. formalized the notion of deniability for AKEs [DGK06]. Since then, a variety of DAKEs have been published [JS08; DKSW09; YYZZ10; CF11; YZ13; Ope13d; WWX14]. Unfortunately, none of these schemes achieve all of our desired properties. pRO-KE [JS08] provides powerful deniability properties, but does not offer forward secrecy. Yao et al. [YYZZ10] construct a protocol that lacks online repudiation, since the MAC keys used by communication partners cannot be computed for simulation. The DAKE from Wen et al. [WWX14] completes in only one round of communication, but requires an expensive designated-verifier proof of knowledge scheme and is only secure against passive adversaries. Implicit DAKE schemes, such as those from Cremers and Feltz [CF11], from Yao and Zhao [YZ13], and 3-DH [Ope13d], all lack online repudiation because they include non-repudiable (and thus non-simulatable) signatures. The scheme introduced as Φ_{dre} by Walfish in his Ph.D. thesis [Wal08], and later reiterated in a publication by Dodis

et al. [DKSW09], does satisfy all of our requirements for a DAKE; we discuss this protocol in greater depth in Section 3.6.

3.3 Overview of Contributions

In this chapter, we advance the state of the art of deniable authenticated key exchanges through the following primary contributions:

1. In Section 3.6.3, we describe an approach that improves the practical performance of the existing Φ_{dre} DAKE in the standard model.
2. In Section 3.7.2, we construct a new interactive DAKE that achieves our strongest notion of deniability with lower latency than Φ_{dre}. We prove its security in Section 3.7.3.
3. In Section 3.8.2, we construct another interactive DAKE that only requires a single round of communication while remaining secure against fully adaptive adversaries. To accomplish this, we admit several obscure attacks that are irrelevant in many environments. We prove the security of this DAKE in Section 3.8.4.
4. In Section 3.8.7, we show how to use our new DAKE in non-interactive environments, with a specific focus on use within TextSecure. In doing so, we partially lose online repudiation.
5. In Section 3.8.8, we conjecture that no DAKE in the TextSecure environment can provide offline repudiation, online repudiation, and forward secrecy simultaneously.
6. In Section 3.8.9, we show how to dramatically improve the performance of our new construction when relaxing the security proof with real-world assumptions.
7. In Section 3.8.10, we show how to use our non-interactive DAKE to bootstrap the Axolotl key ratchet used by TextSecure.

3.4 Cryptographic Preliminaries and Notation

Throughout this chapter, we make use of several specialized cryptosystems when constructing our schemes. This section provides high-level definitions of these underlying cryptosys-

tems. Our schemes can be realized by choosing from the wide range of implementations described in the literature.

3.4.1 Notation

In our constructions, we often need access to randomly generated values. We write $r \xleftarrow{\$} S$ to denote that r is assigned an element from set S selected uniformly at random. For all schemes, we implicitly assume that a security parameter λ is provided to control the security level of the system. Thus, we can write $r \xleftarrow{\$} \{0,1\}^{\mathcal{F}(\lambda)}$ to denote that r is assigned a random binary value with a length controlled by the security parameter. We often initialize cryptosystems and generate keys using this convention, where $\mathcal{F}$ represents a function that adjusts the size of r to be appropriate for the task at hand. As a convenience, we abuse notation by omitting $\mathcal{F}$; we implicitly assume that $r \xleftarrow{\$} \{0,1\}^{\lambda}$ produces a random binary value appropriate to the task at hand, with a security level controlled by λ, even when the length of r might not necessarily be λ in practice.

We denote concatenation of values with the $\|$ operator. As a convenience, we assume that concatenated values are always of a fixed length, allowing anyone examining a concatenated message to unambiguously extract its constituent values. If this is not feasible in practice, an implementation can instead include length prefixes when performing the concatenation.

3.4.2 Digital Signatures

Some of our protocols make use of traditional digital signature schemes, as famously defined by Diffie and Hellman [DH76]. A digital signature scheme consists of the following functions:

- **SigGen**(r): a key generation function. SigGen produces a key pair (pk, sk) for use with the scheme. r is a seed for the algorithm; for any value of r, all invocations of SigGen(r) return the same pair (pk, sk). As a notional convenience, we sometimes omit the parameter to denote that SigGen is called with a fresh value $r \xleftarrow{\$} \{0,1\}^{\lambda}$.
- **Sig**(pk, sk, m, r): a signing function. Sig produces a signature σ for a message m using key pair (pk, sk). r controls the randomization of the output. For any tuple (pk, sk, m, r), Sig returns the same signature across invocations. The scheme may or

may not return differing signatures for the same (pk, sk, m) when r is changed. If r is omitted, it is assumed that $r \xleftarrow{\$} \{0,1\}^\lambda$ is used.

- **Vrf**(pk, σ, m): a verification function. Vrf returns `true` if the signature is valid, and `false` if it is not. If σ was legitimately computed using Sig and pk, then the signature is valid (*correctness*). If sk was never used to legitimately sign m, and sk has not been compromised, then the signature is invalid with overwhelming probability (*soundness*). All calls to Vrf with the same parameters return the same result (*consistency*). In all other cases, no guarantees are made about the output of Vrf.

Our security requirements for digital signatures are based on the idealized model defined by Canetti [Can04], and thus are fairly weak compared to common security notions. Specifically, our only requirements for a "secure" signature scheme are completeness, soundness, and consistency. Our protocols tolerate digital signature schemes even if they exhibit any of the following properties:

- **Incorrect verification keys**: if a signature is verified using a key pk other than the one used to compute the signature, the verification result is adversarially controlled.
- **Public signature malleability**: an adversary given only pk, σ, and m, where σ is a legitimately produced signature for m, can produce a signature σ' that is also a valid signature for m under the same key pk.
- **Corrupted signature claims**: a corrupted signer can produce a public key pk for which Vrf reports that any signature is valid for any message.

For more details concerning these security properties, the interested reader is referred to Canetti's definition of $\mathcal{F}_{\text{SIG}}$ [Can04].

We primarily make use of digital signatures to bind cryptographic keys for other schemes to a single "master" signing key. Because cryptosystems are often based on different underlying algebraic structures, it is often not possible to use a single key with multiple cryptosystems. Moreover, key reuse is generally discouraged; a protocol that reuses keys for multiple cryptosystems may be vulnerable to attacks that do not affect the constituent cryptosystems in isolation. Instead, different keys can be generated for each required scheme and signed by the single master key. Other users can interpret valid signatures as assertions that the signed keys are owned by the entity in possession of the master signing key.

3.4.3 Public-Key Encryption (PKE)

After performing trust establishment, we often need to transmit secret information that can only be read by a given, verified entity. These transmissions can be protected through the use of public-key encryption (PKE). A public-key cryptosystem consists of the following functions:

- **PKGen**(r): a key generation function. PKGen produces a key pair (pk, sk) for use with the scheme. As in our definition for signature schemes, r represents the seed used to generate the key pair and may be omitted to denote that $r \xleftarrow{\$} \{0,1\}^\lambda$ is used.
- **PKEnc**(pk, m, r): an encryption function. PKEnc encrypts a message m under pk to produce a ciphertext γ. The output of PKEnc is consistent across invocations with the same (pk, m, r) as input and varies when r is changed. Any value of r produces a valid encryption of m. If r is omitted, it is assumed that $r \xleftarrow{\$} \{0,1\}^\lambda$ is used.
- **PKDec**(pk, sk, γ): a decryption function. PKDec uses the pair (pk, sk) to decrypt a ciphertext γ that was encrypted under pk. We require that the scheme satisfies $\text{PKDec}(pk, sk, \text{PKEnc}(pk, m, r)) = m$ for any (pk, sk) produced by PKGen and any m and r (*completeness*). In all other cases, PKDec returns the special value $\perp$ with overwhelming probability.

Our schemes that make use of public-key encryption require the cryptosystem to be indistinguishable under adaptive chosen ciphertext attack (i.e., schemes must be secure under IND-CCA2). This strong security requirement, due to Rackoff and Simon [RS92], is defined in the following way:

IND-CCA2 security: any probabilistic polynomial time (PPT) adversary has at most negligible advantage over random guessing in the following security game:

1. The challenger computes $(pk, sk) \leftarrow \text{PKGen}()$ and sends pk to the adversary, while keeping sk secret.
2. The adversary is allowed to make queries to a decryption oracle $\mathcal{O}$ such that $\mathcal{O}(\phi) = \text{PKDec}(pk, sk, \phi)$. The adversary may perform a polynomially bounded number of encryptions, calls to $\mathcal{O}$, and other operations.
3. The adversary constructs two messages m_1 and m_2 of equal length and sends them to the challenger.

4. The challenger generates $b \xleftarrow{\$} \{0,1\}$ and sends $\gamma \leftarrow \text{PKEnc}(pk, m_b)$ to the adversary.

5. Access to $\mathcal{O}$ is revoked for the adversary and replaced with access to a selective decryption oracle $\mathcal{O}'$, such that $\mathcal{O}'(\phi) = \mathcal{O}(\phi)$ unless $\phi = \gamma$, in which case $\mathcal{O}'(\gamma) = \perp$. The adversary may perform a polynomially bounded number of encryptions, calls to $\mathcal{O}'$, and other operations.

6. The adversary outputs a guess b', and wins if $b' = b$.

3.4.4 Dual-Receiver Encryption (DRE)

It is sometimes desirable to encrypt a message such that it can only be read by either of two named recipients. A naïve approach would be to simply encrypt the message m under two asymmetric encryptions—one for each receiver. The message encrypted for recipient keys pk_1 and pk_2 would contain $\text{PKEnc}(pk_1, m)$ and $\text{PKEnc}(pk_2, m)$. Unfortunately, such a scheme lacks public verifiability; the message received by the two parties may differ.

Dual-receiver encryption (DRE) is a type of specialized cryptosystem that enables publicly verifiable encryptions of messages for two receivers. A DRE scheme consists of the following functions:

- **DRGen**(r): a key generation function. DRGen produces a key pair (pk, sk) for use with the scheme. As in our earlier definitions, r represents the seed used to generate the key pair and may be omitted to denote that $r \xleftarrow{\$} \{0,1\}^\lambda$ is used.

- **DREnc**(pk_1, pk_2, m, r): an encryption function. DREnc encrypts a message m under two public keys pk_1 and pk_2 using the same security parameter λ, producing a ciphertext γ. The output of DREnc is consistent across invocations with the same (pk_1, pk_2, m, r) as input and varies when r is changed. Any value of r produces a valid encryption of m. If r is omitted, it is assumed that $r \xleftarrow{\$} \{0,1\}^\lambda$ is used.

- **DRDec**$(pk_1, pk_2, sk_i, \gamma)$: a decryption function. It uses the pair (pk_i, sk_i), where $i \in \{1,2\}$, to decrypt a ciphertext γ that was encrypted under pk_i. We require that the scheme satisfies $\text{DRDec}(pk_1, pk_2, sk_i, \text{DREnc}(pk_1, pk_2, m, r)) = m$ for any (pk_1, sk_1) and (pk_2, sk_2) produced by DRGen and any m and r (*completeness*). In all other cases, DRDec returns the special value $\perp$ with overwhelming probability.

DRE is particularly useful for the construction of DAKEs that are secure against online judges. If a protocol requires that outgoing messages are encrypted for both the intended

recipient as well as the nominal sender, then this allows misinformants to read messages produced in secret by the judge. Specifically, if the judge insists on generating a message encrypted with DRE on behalf of the misinformant, the misinformant will be able to decrypt the message. This decryption often provides the misinformant with a simulation strategy that would be unavailable if simple PKE was used.

We require DRE schemes to exhibit several strong security properties, as defined by Chow et al. [CFZ14]:

- **Symmetry**: all public keys produced by DRGen may be supplied as either argument to DREnc.
- **Public verifiability**: for a given ciphertext γ, any party with knowledge of pk_1 and pk_2 can verify that there exists m and r such that $\gamma = \text{DREnc}(pk_1, pk_2, m, r)$.
- **Soundness**: for any key pairs (pk_1, sk_1) and (pk_2, sk_2), which may or may not have been generated using DRGen, and any value γ, $\text{DRDec}(pk_1, pk_2, sk_1, \gamma) = \text{DRDec}(pk_1, pk_2, sk_2, \gamma)$.
- **Dual-receiver IND-CCA1 security**: any probabilistic polynomial time (PPT) adversary has at most negligible advantage over random guessing in the following security game:

  1. The challenger computes $(pk_1, sk_1) \leftarrow \text{DRGen}()$ and $(pk_2, sk_2) \leftarrow \text{DRGen}()$, then sends pk_1 and pk_2 to the adversary, while keeping sk_1 and sk_2 secret.
  2. The adversary is allowed to make queries to a decryption oracle $\mathcal{O}$ such that $\mathcal{O}(\phi) = \text{DRDec}(pk_1, pk_2, sk_1, \phi)$. The adversary may perform a polynomially bounded number of encryptions, calls to $\mathcal{O}$, and other operations.
  3. The adversary constructs two messages m_1 and m_2 of equal length and sends them to the challenger.
  4. The challenger generates $b \xleftarrow{\$} \{0, 1\}$ and sends $\gamma \leftarrow \text{DREnc}(pk_1, pk_2, m_b)$ to the adversary.
  5. Access to $\mathcal{O}$ is revoked for the adversary. The adversary may perform a polynomially bounded number of encryptions and other operations.
  6. The adversary outputs a guess b', and wins if $b' = b$.

  We remark that the choice of secret key used by $\mathcal{O}$ to decrypt messages is irrelevant due to the soundness of the scheme.

Chow et al. [CFZ14] also define the notion of *complete non-malleability* for DRE schemes. Their definition is an extension of the corresponding definition for public-key cryptosystems introduced by Fischlin [Fis05] and later extended by Ventre and Visconti [VV08]. A completely non-malleable DRE scheme ensures that ciphertexts are non-malleable even when the adversary is permitted to change the public keys used for encryption (possibly to adversarially generated keys). Our schemes do not require this stronger security notion, thereby allowing us to use more efficient DRE implementations in practice.

3.4.5 Non-Committing Encryption (NCE)

A public-key cryptosystem is called *non-committing* if, in addition to offering the standard functions described in Section 3.4.3, it also offers the ability to produce "rigged" ciphertexts. Specifically, it is possible to construct a ciphertext and public key that can later be decrypted to any plaintext by using secret information to construct a corresponding secret key. This notion, first proposed by Canetti et al. [CFGN96], is useful for proving the security of protocols against adversaries that can adaptively corrupt parties. We later revisit the usefulness of NCE with respect to adaptive corruptions when we discuss the security model for our protocols.

A non-committing cryptosystem consists of the following functions:

- **NCGen**(r): a key generation function. NCGen produces a key pair (pk, sk) for use with the scheme. As in our earlier definitions, r represents the seed used to generate the key pair and may be omitted to denote that $r \xleftarrow{\$} \{0,1\}^\lambda$ is used.
- **NCEnc**(pk, m, r): an encryption function. NCEnc encrypts a message m under pk to produce a ciphertext γ. The output of NCEnc is consistent across invocations with the same (pk, m, r) as input, but may or may not vary when r is changed. Any value of r produces a valid encryption of m. If r is omitted, it is assumed that $r \xleftarrow{\$} \{0,1\}^\lambda$ is used.
- **NCDec**(pk, sk, γ): a decryption function. NCDec uses the pair (pk, sk) to decrypt a ciphertext γ that was encrypted under pk. We require that the scheme satisfies $\text{NCDec}(pk, sk, \text{NCEnc}(pk, m, r)) = m$ for any (pk, sk) produced by NCGen and any m and r (*completeness*). In all other cases, NCDec returns the special value $\perp$ with overwhelming probability.
- **NCSim**(r): a rigged ciphertext generation function. NCSim produces a public key pk, a ciphertext γ, and some auxiliary information α. pk is indistinguishable from a

"normal" public key generated by NCGen, and γ is indistinguishable from a "normal" ciphertext generated by NCEnc. Like NCGen, r controls the pairs that are generated and can be omitted to denote a random seed is used.

- **NCEqv**(pk, γ, α, m): an equivocation function. If NCEqv is called with (pk, γ, α) produced by NCSim, it generates values sk, r^*, and r^{NCE} such that NCGen$(r^*) = (pk, sk)$ and NCEnc$(pk, m, r^{NCE}) = \gamma$. In other words, it uses α to generate a secret key sk corresponding to pk such that the previously generated γ decrypts to m using sk. All of this is accomplished while making key pair (pk, sk) appear as though it were honestly generated using NCGen, and γ as though it was honestly generated using NCEnc.

There are several existing NCE constructions [CFGN96; DN00a; CDMW09; ZANS12]. However, all existing constructions are considerably more expensive (in terms of computation time or communication size) than the other primitives used in our protocols.

3.4.6 Ring Signatures

Ring signatures, originally proposed by Rivest et al. [RST01], are a specialized type of digital signature scheme. A ring signature is verifiably produced by a member of a given set (known as the *ring*), but the exact identity of the signer cannot be determined. Ring signatures are useful for constructing DAKEs because they allow parties to authenticate to each other in a non-transferable manner. Concretely, a ring signature with a ring containing both the sender and receiver of a message proves to the receiver that the message was signed by the sender (since the receiver knows that they did not sign the message themselves). However, the receiver cannot convince anyone else of this fact—since they are a member of the ring, the receiver could have produced the message. We later make use of this property, along with the ability to construct rings with more than two members, to construct our DAKEs.

A ring signature scheme consists of the following functions:

- **RSGen**(r): a key generation function. RSGen produces a key pair (pk, sk) for use with the scheme. As in our earlier definitions, r represents the seed used to generate the key pair and may be omitted to denote that $r \xleftarrow{\$} \{0,1\}^\lambda$ is used.
- **RSig**(pk, sk, R, m, r): a signing function. RSig produces a signature σ for a message m. The ring R is a set of n public keys $\{pk_1, pk_2, \ldots, pk_n\}$ that could possibly have

produced the signature. It is required that $n > 1$, (pk, sk) was generated with RSGen, and $pk \in R$. r controls the randomization of the output. For any (pk, sk, R, m, r), RSig returns the same signature across invocations. The scheme returns differing signatures for the same (pk, sk, R, m) when r is changed. If r is omitted, it is assumed that $r \xleftarrow{\$} \{0,1\}^\lambda$ is used.

- **RVrf**(R, σ, m): a verification function. RVrf returns `true` if the signature is valid, and `false` if it is not. Minimally, it is required that $\text{RVrf}(R, \text{RSig}(pk, sk, R, m, r), m) =$ `true` for any valid inputs (*correctness*).

Over the years, different authors have proposed varied definitions of security properties for ring signatures. We make use of the definitions introduced by Bender et al. [BKM06], which classify ring signature schemes based on their anonymity guarantees and signature forgeability. In this work, we require the use of ring signatures that achieve the following strong security properties:

- **Anonymity against full key exposure**: an adversary cannot determine which honest party produced a signature, even when given access to a signing oracle, and the secret keys for all parties after the challenge signature has been generated.
- **Unforgeability with respect to insider corruption**: an adversary can never produce a valid signature for a ring that does not include a corrupted party, even when given access to a signing oracle and the capability to adaptively corrupt parties (thereby compromising their secret keys).

3.5 The GUC Framework

3.5.1 Universal Composability

When designing a new cryptographic protocol, we would like to have some guarantees about its security properties. The standard approach to this problem is to create a mathematical model of the real-world environment, and prove that the protocol is secure in this model with respect to some security definition. In the past, this approach has led to problems with modeling complex protocols; often, the definition of a protocol makes use of other protocols as subroutines (e.g., a DAKE might make use of a digital signature scheme). When constructing a composite scheme of this type, its security model and security definition must account not only for the overall scheme, but also for its constituent

parts. This approach quickly becomes unmanageable for complex systems such as secure messaging protocols, which may consist of dozens of cryptographic primitives and must guard against a wide assortment of adversaries.

To address these concerns, Canetti introduced the notion of Universal Composability (UC) [Can01]. The UC framework provides a means for proving that a protocol retains its security properties even when used as part of a larger scheme with other arbitrary protocols being executed concurrently. This approach is particularly well suited to constructing complex schemes for use on the Internet. While this chapter assumes some familiarity with UC, we briefly summarize the framework in this section.

The general idea behind the UC framework is to prove that a real protocol behaves identically to an idealized protocol with well-defined security properties. Once this equivalence has been established, more complex schemes can be constructed that make use of the idealized protocol as a subroutine. When implementing a scheme, the overall security is guaranteed to be preserved when replacing the idealized subroutine with any real protocol proven to behave identically.

Ideal protocols in the UC framework consist of a set of parties interacting with a trusted central authority over secure connections. In an ideal protocol $\text{IDEAL}_{\mathcal{F}}$, the trusted authority executes some code $\mathcal{F}$ referred to as the *ideal functionality*. $\mathcal{F}$ defines the types of messages that the protocol participants can send to the authority, the computations that it performs, and the responses that are sent to participants. In addition to the main parties of the protocol, $\text{IDEAL}_{\mathcal{F}}$ also models the existence of an adversary $\mathcal{S}$, which is able to interact with the trusted authority in a manner prescribed by $\mathcal{F}$. Each invocation of $\text{IDEAL}_{\mathcal{F}}$ represents a single session of the protocol. The main parties of the protocol begin the session with some input values that they send along their secure channels to the trusted authority. The trusted authority performs some computation on the inputs and sends the result along its secure channels to the main parties, which then output these values.

"Real" protocols (e.g., those designed to take place between Internet users) are also modeled in the UC framework. These protocols consist of messages exchanged between a set of parties in the presence of an adversary $\mathcal{A}$. We stress that, unlike ideal protocols, these real protocols do not generally make use of a trusted party.

The ultimate goal of a security proof in the UC framework is to demonstrate that a real protocol under consideration somehow behaves identically to an ideal protocol, permitting the ideal protocol to be used in the construction of more complex schemes without worrying about potentially unexpected interactions. Intuitively, we can achieve this goal if the environment surrounding a challenge protocol does not behave significantly differently

when interacting with the real or ideal protocols under any given attack. The *environment* consists of the collection of other protocols running in the system; these other protocols may influence the input of the challenge protocol, or change their behavior based on the output of the challenge protocol. We write $\text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}$ to denote the distribution of outputs of an environment $\mathcal{Z}$ interacting with a protocol π involving an adversary $\mathcal{A}$. With this definition, we now reiterate Canetti's definition of protocol indistinguishability [Can01]:

Definition 3.1 (*UC-emulation*)

> A protocol π_1 UC-emulates protocol π_2 if for any adversary $\mathcal{A}$ there exists an adversary $\mathcal{S}$ such that, for any environment $\mathcal{Z}$, the ensembles $\text{EXEC}_{\pi_1,\mathcal{A},\mathcal{Z}}$ and $\text{EXEC}_{\pi_2,\mathcal{S},\mathcal{Z}}$ are indistinguishable.

When real protocol π UC-emulates ideal protocol $\text{IDEAL}_{\mathcal{F}}$, we say π *UC-realizes* $\text{IDEAL}_{\mathcal{F}}$.

Typically, it is not possible for a real protocol to UC-realize an ideal protocol $\text{IDEAL}_{\mathcal{F}}$ unless $\mathcal{F}$ permits some interactions with the adversary $\mathcal{S}$. Intuitively, information released by $\mathcal{F}$ to $\mathcal{S}$ represents information that is leaked by the real protocol. Commands sent to $\mathcal{F}$ by $\mathcal{S}$ represent the possible influence of an adversary on the execution of the real protocol.

The computation model of the UC framework is constructed in terms of interactive Turing machines (ITMs). An ITM represents the instructions of a Turing machine with access to three tapes: an input tape, where "secure" messages are queued for delivery to the machine; a communication tape, where "insecure" messages are written from other machines; and a subroutine output tape. Callers write parameters to the input tapes of subroutines; the subroutines can later return output by writing to the subroutine output tape of the caller. A running instance of an ITM is called an interactive Turing instance (ITI). Writing to the input or subroutine output tape of an ITI is a privileged operation, and represents communication over a secure channel. In contrast, messages sent on communication tapes are unauthenticated and visible to the adversary. When an ITI sends a secure message to another ITI, it includes the ITM (i.e., the code) of the target ITI as part of its message. If the ITI was not already running, it is started using the given code. An ITM can contain a "halt" instruction; an ITI that encounters a halt instruction stops executing code and thus does not process any data written to its tapes. The formal definition of the UC framework contains substantially more detail about the computation model; the interested reader is referred to the original paper for in-depth formalizations [Can01].

The first ITI to be launched as part of a challenge protocol is the distinguishing environment $\mathcal{Z}$. This machine is permitted to launch one ITI representing the adversary $\mathcal{A}$ for the challenge protocol, as well as ITIs for each main party of the protocol. $\mathcal{Z}$ is not permitted to read the code associated with the ITIs that it launches, as this would make

it trivial to distinguish between challenge protocols. $\mathcal{Z}$ provides all inputs to the protocol by writing to the input tapes of the main parties. The main parties of the protocol return their outputs to $\mathcal{Z}$ by writing to its subroutine output tape. $\mathcal{Z}$ is also able to communicate with $\mathcal{A}$ by writing to its input tape; $\mathcal{A}$ can return results by writing to $\mathcal{Z}$'s subroutine output tape. $\mathcal{Z}$ is not permitted to view transmissions on the communication tapes, or to interact with subroutines invoked by any of the parties. In contrast, the adversary is granted widespread power by default. $\mathcal{A}$ can view, modify, drop, and generate messages transmitted on communication tapes between the main parties. $\mathcal{A}$ can also send a special corruption message to the main parties of the protocol. When a party receives a corruption message, it notifies $\mathcal{Z}$ that it has been corrupted, and then sends its current and historical states to the adversary (including all messages sent from and received by the corrupted party). $\mathcal{A}$ is then considered to have full control of the corrupted party, and may send messages to other parties using the identity of the corrupted party. Any messages received by corrupted parties are automatically forwarded to $\mathcal{A}$.

A common situation that must be modeled in ideal functionalities is the notion of *delayed output*. This represents the situation in real protocols where the adversary delays or drops the transmission of a message on the network (or, in the context of the UC framework, the message written to a communication tape). When we write that the ideal functionality $\mathcal{F}$ sends a delayed output message m to a party P, we implicitly give the adversary $\mathcal{S}$ the authority to delay or drop the message. $\mathcal{F}$ sends a message to $\mathcal{S}$ stating that a message is about to be sent to P. If m is a *public delayed output*, then this communication also provides the contents of m to $\mathcal{S}$. When $\mathcal{F}$ receives a response from $\mathcal{S}$, it actually sends m to P. $\mathcal{S}$ can effectively drop the message by never providing permission to transmit m, or it may delay transmission by waiting before providing permission. If $\mathcal{F}$ halts, then any outstanding delayed messages are never delivered.

If a real protocol only realizes an ideal protocol under certain attacks, then the security model can place restrictions on the capability of the adversary. Some common restrictions include limits on the timing of corruptions and the nature of revealed state information. If $\mathcal{A}$ can corrupt any party at any time, then the security model is said to permit *adaptive corruptions*. If $\mathcal{A}$ can corrupt parties only before the protocol session begins, the model is said to permit only *static corruptions*. If corruptions are permitted only at certain designated times, then the model is said to permit only *semi-adaptive corruptions*. In the default model, parties reveal their entire state history to $\mathcal{A}$ upon corruption; this is called the *non-erasure* model. The *erasure* model permits parties to securely erase state so that it is not revealed during subsequent corruptions.

To achieve composability, a protocol π can be permitted to invoke one or more instances of another functionality $\mathcal{F}'$. Such a protocol is referred to as a $\mathcal{F}'$-hybrid protocol, and its security is examined in the $\mathcal{F}'$-hybrid model.

3.5.2 Generalized UC (GUC)

While the basic UC framework can provide excellent security guarantees for many protocols, it can only be used when a protocol is "self contained". Specifically, UC assumes that a protocol does not have access to an ITI that belongs to another session. Unfortunately, this assumption makes it difficult to model relatively common situations in which information persists between sessions, as this information could potentially be used by $\mathcal{Z}$ to distinguish between challenge protocols. Protocols that store long-term keys in a public key directory, or that share data in a common reference string (CRS), are examples of situations that are difficult to model. A simple way to adapt the UC framework to these situations is to consider all sessions of a protocol to be part of a single UC "session". Such an approach negates many advantages of using UC in the first place—any higher-level protocol that makes use of a scheme with such a proof must itself be proven secure in a multi-session environment.

To combat these issues, Canetti and Rabin introduced the notion of universal composition with "Joint State" (JUC) [CR03]. JUC modifies UC to include the notion of multi-session protocols. When a scheme invokes a functionality several times, these subroutines can be combined into a single multi-session protocol in a way that preserves the expected security properties. While the JUC framework achieves its goal of modeling protocols with shared state while maintaining the usability of the UC framework, it does not provide security guarantees against *adaptive chosen protocol* attacks. If an adversary introduces a new malicious protocol that makes use of the shared state, JUC provides no security guarantees for honest parties who choose (perhaps unknowingly) to make use of the malicious protocol.

In response, Walfish introduced the Generalized UC (GUC) framework [Wal08]. GUC extends UC to allow multiple concurrent protocol sessions with shared information. It also allows the adversary and environment to access the shared information. Additionally, the environment is permitted to execute arbitrary other protocols. Despite this additional distinguishing power, GUC can still provide the usual UC security guarantees. When a real protocol π emulates an ideal protocol $\text{IDEAL}_{\mathcal{F}}$ within the GUC framework, we say that π *GUC-realizes* $\text{IDEAL}_{\mathcal{F}}$.

GUC models shared information between UC sessions through the use of "shared functionalities". A shared functionality is a typical UC functionality that is executed by an ITI that persists between protocol sessions. For example, $\bar{\mathcal{G}}_{krk}^{\mathcal{F}}$ (key registration with knowledge) represents a key directory functionality that models a real PKI system; public keys for parties in IDEAL$_{\mathcal{F}}$ are available from $\bar{\mathcal{G}}_{krk}^{\mathcal{F}}$ upon request, and $\mathcal{F}$ (or corrupted parties) are additionally permitted to retrieve the associated secret keys. If a protocol π does not share state with any other protocol sessions, with the exception of the shared functionality $\bar{\mathcal{G}}$, then π is called $\bar{\mathcal{G}}$-*subroutine respecting*.

Many simulation strategies that are secure in the basic UC framework are no longer viable in the GUC framework because $\mathcal{Z}$ is given access to shared functionalities. For example, a simulator in the UC framework could produce a digital signature for a party P by simulating P's generation of long-term key pair (PK'_P, SK'_P) and using SK'_P to sign a message. In the GUC framework, $\mathcal{Z}$ would easily be able to distinguish such a transcript from a real one by querying $\bar{\mathcal{G}}_{krk}^{\mathcal{F}}$ for PK_P, the true long-term public key associated with P, and discovering that $PK_P \neq PK'_P$.

Since GUC adds a lot of additional complexity over UC in order to more accurately model real-world systems, security proofs written directly within the GUC model are more difficult to formulate. For this reason, Walfish also introduced the External-subroutine UC (EUC) framework [Wal08]. EUC is identical to the basic UC framework, except it allows the protocol, adversary, and environment to access a single shared functionality $\bar{\mathcal{G}}$. Such an environment is called $\bar{\mathcal{G}}$-*externally constrained*. When a real protocol π emulates a protocol ϕ within a $\bar{\mathcal{G}}$-externally constrained environment, we say that π $\bar{\mathcal{G}}$-*EUC-emulates* ϕ. We can now restate a surprising theorem proved by Walfish [Wal08]:

Theorem 3.1 (*EUC-emulation is GUC-emulation*)

> Let π be any protocol that is $\bar{\mathcal{G}}$-subroutine respecting for shared functionality $\bar{\mathcal{G}}$. Then protocol π GUC-emulates a protocol ϕ if and only if protocol π $\bar{\mathcal{G}}$-*EUC-emulates* ϕ.

Due to Theorem 3.1, if a protocol π is $\bar{\mathcal{G}}$-subroutine respecting, then a proof of security for π in the EUC framework provides all of the security guarantees of an equivalent proof in the GUC framework. This result greatly simplifies the task of using the GUC framework, since security proofs within EUC are only marginally more complex than proofs performed in the basic UC framework.

The UC framework and its derivatives are particularly useful for discussing deniable protocols. If a real protocol is shown to realize an ideal protocol, then this means that the protocol is fully simulatable in all environments covered by the security model. This is due to the fact that any attack launched against a real protocol session is indistinguishable from

a simulated attack on the publicly known ideal functionality, and thus an attacker cannot learn any more information from attacking a real session than they could from simulating their attack. In this sense, the real protocol is shown to be as deniable as the ideal protocol. In relating these notions to the discussion of deniability in Section 3.1, $\mathcal{Z}$ represents the judge and $\mathcal{A}$ the informant (with $\mathcal{S}$ representing the misinformant). If a real protocol π GUC-realizes an ideal protocol $\text{IDEAL}_{\mathcal{F}}$, then the judge can never distinguish between the actions of an informant $\mathcal{A}$ interacting with a real session of π and a misinformant simulating $\mathcal{S}$ interacting with $\text{IDEAL}_{\mathcal{F}}$. If $\mathcal{Z}$ provides instructions to the adversary during the protocol execution, then $\mathcal{Z}$ is effectively an online judge.

3.6 The Walfish Protocol

In his thesis, Walfish [Wal08] introduced Φ_{dre}.[2] As discussed in Section 3.2, Φ_{dre} is the only previously defined DAKE of which we are aware that provides deniability against both online and offline judges while simultaneously providing forward secrecy. Walfish proved the security properties of Φ_{dre} in the GUC framework. We briefly discuss the ideal functionality used in this proof, and then present the real protocol.

3.6.1 Ideal Functionality $\mathcal{F}_{keia}^{\text{IncProc}}$

In order to prove the security of Φ_{dre}, Walfish modeled the notion of a key exchange protocol within the GUC framework. A key exchange takes place between two parties: an initiator I, and a responder R.[3] Trust establishment is assumed to take place through a PKI, and thus the key exchange functionality takes place in the $\bar{\mathcal{G}}_{krk}$-hybrid model. I and R are assumed to have set up keys in the PKI before the session begins. A simple way to model such a key exchange is with ideal functionality $\mathcal{F}_{ke}$. $\mathcal{F}_{ke}$ waits to receive a `key-exchange` message from both I and R. $\mathcal{S}$ is then given an opportunity to attempt to control the key produced by the exchange through a `set-key` message. If $\mathcal{S}$ has corrupted R and I has started a key exchange, then $\mathcal{S}$ can cause both parties to output any key k'. Otherwise, $\mathcal{F}_{ke}$ generates a fresh shared secret key k and sends it to each party (if they previously sent a `key-exchange` message) through their secure channel.

[2] The protocol was originally introduced in Walfish's Ph.D. thesis [Wal08]. It was later restated in a publication by Dodis et al. [DKSW09].

[3] The original protocol definition refers to these parties as a sender S and a receiver R. Here, we denote them as I and R for clarity and to be consistent with other referenced schemes.

Algorithm 1 Ideal functionality $\mathcal{F}_{keia}^{\text{IncProc}}$ (adapted from original [Wal08])

on receipt of $(\texttt{key-exchange}, sid, I, R, SK_S)$ **from** I:
  if (I is "active" || I is "aborted") **return**
  Mark I as "active"
  Send public delayed $(\texttt{key-exchange}, sid, I, R)$ to R

on receipt of $(\texttt{key-exchange}, sid, I, R, SK_R)$ **from** R:
  if (R is "active') **return**
  Mark R as "active"
  Send public delayed $(\texttt{active}, sid, I, R)$ to I

on receipt of $(\texttt{set-key}, sid, I, R, k')$ **from** $\mathcal{S}$:
  if (R is corrupt **&&** I is "active") {
    Send $(\texttt{set-key}, sid, I, R, k')$ to I and R
  } **else if** (R is honest **&&** R is "active") {
    $k \xleftarrow{\$} \{0,1\}^\lambda$
    Send $(\texttt{set-key}, sid, I, R, k)$ to R
    if (I is "active") Send delayed $(\texttt{set-key}, sid, I, R, k)$ to I
  }
  Halt

on receipt of $(\texttt{abort}, sid, I, R)$ **from** $\mathcal{S}$:
  if (I is "active") {
    Mark I as "aborted"
    Send delayed $(\texttt{abort}, sid, I, R)$ to I
  }
  if (R is "active") Send delayed $(\texttt{abort}, sid, I, R)$ to R

on receipt of $(\texttt{incriminate}, sid, I)$ **from** $\mathcal{S}$:
  if (an `incriminate` message has already been received) **return**
  if (I is "aborted" **&&** I is honest) {
    Execute IncProc$(sid, I, R, PK_I, PK_R, SK_I)$
  }

Unfortunately, Walfish proved that $\mathcal{F}_{ke}$ cannot be realized in the presence of adaptive corruptions [Wal08]. Instead, we must relax the functionality by leaking some additional information to the adversary. $\mathcal{S}$ is additionally granted the ability to cause a key exchange session to abort. When a session aborts, some incriminating, non-simulatable information is leaked about one of the parties. Since the exact nature of the leaked information depends on the real protocol being used, the functionality is parameterized with an ITM IncProc that handles the details of the incriminating leak. The real-world implication of IncProc is that an adversary can generally break the deniability of protocols by somehow causing the key exchange to fail. For the remainder of this chapter, whenever we introduce a DAKE, we discuss the real-world implications of the IncProc definition used in its security proof. When $\mathcal{F}_{ke}$ is parameterized with IncProc, the resulting functionality is called $\mathcal{F}_{keia}^{\text{IncProc}}$ (key exchange with incriminating abort). We restate Walfish's definition of $\mathcal{F}_{keia}^{\text{IncProc}}$ in Algorithm 1.

As with $\mathcal{F}_{ke}$, $\mathcal{F}_{keia}^{\text{IncProc}}$ begins by waiting for I and R to send `key-exchange` messages, indicating their willingness to perform the key exchange. When $\mathcal{S}$ sends a `set-key` message, either a fresh key k or an adversarial key k' is distributed, based on the corruptions made by $\mathcal{S}$. In addition, $\mathcal{F}_{keia}^{\text{IncProc}}$ allows $\mathcal{S}$ to send an `abort` message to terminate the protocol. An abort causes delayed notifications to be sent to all active parties, but these can be withheld by $\mathcal{S}$ if it chooses to do so. $\mathcal{S}$ can cause R to output a key after an abort has occurred, but it cannot cause I to output a key after an abort. Finally, $\mathcal{S}$ is also allowed to send an `incriminate` message to $\mathcal{F}_{keia}^{\text{IncProc}}$ if the protocol has been aborted. Upon receipt of this message, $\mathcal{F}_{keia}^{\text{IncProc}}$ invokes a new ITI that executes IncProc. The intent of IncProc is to allow $\mathcal{S}$ to provide a partial transcript from a real protocol execution, to which IncProc will respond with an incriminating message constructed using I's secret key SK_I.

Proofs of realization of $\mathcal{F}_{keia}^{\text{IncProc}}$ take place in the $\bar{\mathcal{G}}_{krk}$-hybrid model. Recall that $\bar{\mathcal{G}}_{krk}$ (key registration with knowledge) is a shared functionality that models a PKI. Any party P can register for a key pair with $\bar{\mathcal{G}}_{krk}$. $\bar{\mathcal{G}}_{krk}$ generates the requested key pair directly, thereby modeling a PKI that requires proof of knowledge of secret keys. A corrupted party can override this by providing its own key pair for storage. PK_P, the public key for a party P, can be retrieved from $\bar{\mathcal{G}}_{krk}$ by any machine. $\bar{\mathcal{G}}_{krk}^{\mathcal{F}}$ also allows SK_P, the secret key for party P, to be retrieved by an honest ITI executing $\mathcal{F}$ or by P if it has been corrupted.

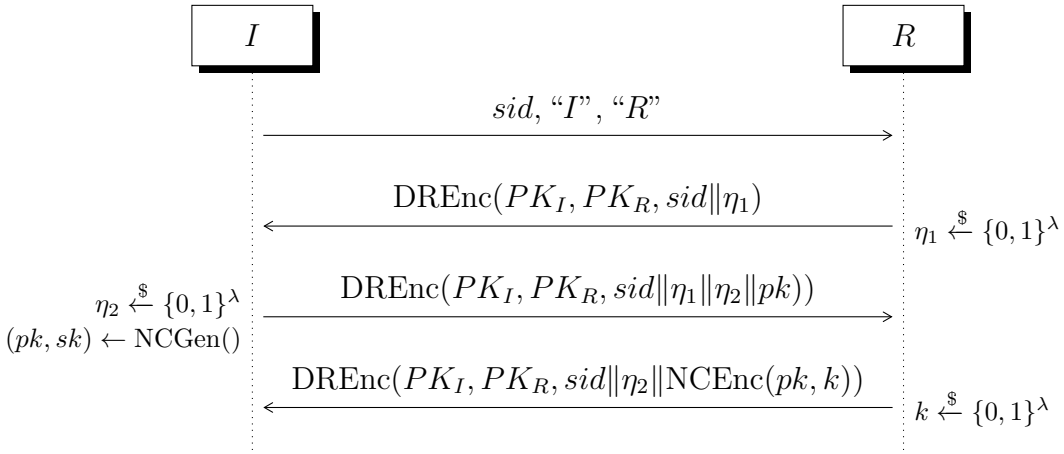


Figure 3.1: Real protocol Φ_{dre}. The shared secret is k.

3.6.2 Real Protocol Φ_{dre}

Φ_{dre}, depicted in Figure 3.1, is a two-round interactive DAKE.[4] Walfish has previously shown that Φ_{dre} GUC-realizes $\mathcal{F}_{keia}^{\text{IncProc}}$ in the $\bar{\mathcal{G}}_{krk}^{\Phi_{dre}}$-hybrid model with semi-adaptive security (i.e., the adversary $\mathcal{A}$ may not corrupt I or R while the protocol is executing) [Wal08].

Φ_{dre} leverages the security of dual-receiver encryption (see Section 3.4.4) to provide its deniability guarantees. Each message sent between I and R (other than the introductory message asserting identities) is encrypted using DRE under the public keys of I and R. This guarantees that all three messages can be read by both I and R, but nobody else (unless either I or R have been corrupted). To provide authentication, the core of the protocol involves each party echoing a nonce generated by their partner, thereby proving that they can decrypt the communications. Since only I and R can decrypt the DRE, and each party knows that they did not produce the response to their nonce themselves, this provides non-transferable authentication. In addition, I includes in its encrypted message an ephemeral public key pk for an NCE scheme (see Section 3.4.5). R generates the shared secret for the session, and encrypts it using pk as part of its final message to I.

It is trivial for anyone with access to the PKI to forge protocol transcripts between any two parties, even without access to any long-term secret keys. An offline forger of

[4] In practice, the protocol can be collapsed into three flows by having the party denoted as R in Figure 3.1 send the session identifier and party names. In such a case, it would be prudent to swap the names of the parties, since the party sending the session identifier is the protocol initiator. In this chapter, we continue to discuss Φ_{dre} as defined in Figure 3.1, but we consider the protocol to consist of only three flows when evaluating its performance.

this type merely needs to simulate both participants; although it cannot decrypt the DRE layer itself, the simulator already knows the contents of all messages. Deniability against online judges is provided because each party can simulate the behavior of the other; the DRE scheme allows misinformants to read the contents of any messages that the judge produces on their behalf. The use of NCE is only needed in the non-erasure model with semi-adaptive corruptions; if erasures are allowed or corruptions are only static, then a standard PKE scheme can be used instead. Complete details of the simulation steps can be derived from the proof of security given by Walfish [Wal08].

If the protocol aborts, the definition of IncProc for Φ_{dre} releases incriminating information demonstrating that I was attempting to communicate with R. If $\mathcal{A}$ modifies the initial message from R to I to include a different nonce η_1', then I will respond with $\text{DREnc}(PK_I, PK_R, sid \| \eta_1' \| \eta_2 \| pk)$. In practice, the use of IncProc admits an attack in which an online judge can determine if an adversary is simulating the conversation. If Justin, the judge, instructs Mallory, the adversary, to modify the first flow from R to I to an encryption of nonce η_1' known only to Justin, then Mallory will need to invoke IncProc to simulate I's response to R. This requires Mallory to actually interact with the real I (i.e., Mallory is a legitimate informant), or to compromise SK_I or SK_R without the knowledge of Justin (something disallowed by the GUC framework). If Mallory simulates a response without IncProc, then Justin can later instruct Mallory to corrupt a party to recover their secret key. Justin can then use this secret key to decrypt the simulated response, and determine that it did not include η_1'.

3.6.3 An Efficient Instantiation with Interactive DRE

In order to implement Φ_{dre}, we must select a DRE scheme to use. While any of the schemes referenced in Section 3.4.4 could be used, nearly all of them require use of the random oracle model. The DRE construction of Chow et al. [CFZ14] is relatively efficient and is secure in the standard model, but it is still expensive compared to simple encryption schemes. Alongside the original definition of Φ_{dre}, Walfish describes a construction of a generic DRE scheme [Wal08]. This scheme involves encrypting the plaintext twice using a PKE scheme with IND-CCA2 security, and then providing two non-interactive zero-knowledge proofs of knowledge (NIZKPK) demonstrating that the encrypted ciphertexts are equal. Unfortunately, NIZKPK schemes are either highly inefficient or require random oracles. In this section, we describe a new DRE construction that can improve the practical performance of Φ_{dre} while still maintaining all of its security properties in the standard model.

We begin by making an important observation about Φ_{dre}: it is an interactive protocol that takes place between two known parties. While DRE in general is a non-interactive protocol, allowing the DRE scheme to require interactivity does not negatively impact the general properties of the overall scheme. We are able to do this because for each encryption of a message, the only party that will need to decrypt the message is available for interactive communications. In exchange for this concession, we can construct an efficient DRE scheme in the standard model.

Our basic approach to the construction is similar to Walfish's general DRE scheme, but we make use of an interactive zero-knowledge proof of knowledge (ZKPK) scheme instead. While we will only describe one possible instantiation, any PKE scheme with IND-CCA2 security can be combined with any interactive ZKPK of plaintext equality. This DRE remains "publicly verifiable" in the sense that the ZKPK verifier can verify the correctness of the ciphertexts even if they do not know any secret keys; this is sufficient for use in Φ_{dre}. As a PKE scheme, we make use of the cryptosystem published by Cramer and Shoup [CS98]. The Cramer-Shoup scheme provides IND-CCA2 security in the standard model with only the DDH assumption. To prove that the two ciphertexts contain identical messages and are of a valid format, we use a Σ ZKPK of the kind described by Schnorr [Sch91]. The resulting scheme is secure with only the DDH assumption for the underlying Cramar-Shoup group, and consists of the following functions:

- **DRGen**(r): keys are generated as in the Cramer-Shoup scheme [CS98]. The resulting public key for a user consists of a group description (G, q, g_1, g_2) and values $c = g_1^{x_1} g_2^{x_2}$, $d = g_1^{y_1}$, and $h = g_1^z$. The corresponding secret key is (x_1, x_2, y_1, y_2, z).

- **DREnc**(pk_1, pk_2, m, r): m is encrypted twice using Cramer-Shoup (once for each public key), and a ZKPK of plaintext equality is produced. r is interpreted as $r = k_1 \| k_2$ and is used for the randomization of the encryptions of m. Given public key $pk_i = (G_i, q_i, g_{1i}, g_{2i}, c_i, d_i, h_i)$, the resulting encryptions consist of $u_{1i} = g_{1i}^{k_i}$, $u_{2i} = g_{2i}^{k_i}$, $e_i = h_i^{k_i} m$, and $v_i = c_i^{k_i} d_i^{k_i \alpha_i}$ for $i \in 1, 2$ and $\alpha_i = H(u_{1i} \| u_{2i} \| e_i)$ where H is a collision-resistant hash function.

  The result also includes an interactive ZKPK that proceeds between the prover $\mathcal{P}$ (the party calling DREnc) and the verifier $\mathcal{V}$ (the party that will call DRDec) as follows:

  1. $\mathcal{P}$ generates random values $m_i \in [0, q_i)$ for $i \in \{1, 2\}$. $\mathcal{P}$ then computes $T_{1i} = g_{1i}^{m_i}$, $T_{2i} = g_{2i}^{m_i}$, $T_{3i} = (c_i d_i^{\alpha_i})^{m_i}$, and $T_4 = \frac{h_1^{m_1}}{h_2^{m_2}}$, and sends these values to $\mathcal{V}$.
  2. $\mathcal{V}$ generates random value L and sends it to $\mathcal{P}$.

3. $\mathcal{P}$ computes $n_i = m_i - Lk_i \pmod{q_i}$ for $i \in \{1, 2\}$ and sends these values to $\mathcal{V}$.
4. $\mathcal{V}$ accepts the encryption as valid if the following equalities hold for $i \in \{1, 2\}$: $T_{1i} \stackrel{?}{=} g_{1i}^{n_i} u_{1i}^L$, $T_{2i} \stackrel{?}{=} g_{2i}^{n_i} u_{2i}^L$, $T_{3i} \stackrel{?}{=} (c_i d_i^{\alpha_i})^{n_i} v_i^L$, and $T_4 \stackrel{?}{=} \frac{h_1^{n_1}}{h_2^{n_2}} (\frac{e_1}{e_2})^L$.

- **DRDec**$(pk_1, pk_2, sk_i, \gamma)$: γ is parsed to locate the encryption for pk_i, and decryption proceeds as in Cramer-Shoup. Let $sk_i = (x_{1i}, x_{2i}, y_{1i}, y_{2i}, z_i)$. At this point, the recipient of γ has already verified that the ciphertexts are of the correct form and that they contain encryptions of the same message as a result of the interactive ZKPK. In addition, the recipient computes $\alpha_i = H(u_{1i} \| u_{2i} \| e_i)$ and then verifies that $u_{1i}^{x_{1i}} u_{2i}^{x_{2i}} (u_{1i}^{y_{1i}} u_{2i}^{y_{2i}})^{\alpha_i} \stackrel{?}{=} v_i$. The message m is recovered using $m = \frac{e_i}{u_{1i}^{z_i}}$.

The resulting protocol consists of 9 messages (plus an additional message for the introductory identity assertions). This instantiation of Φ_{dre} is very efficient compared to implementations using non-interactive DRE in the standard model, which typically require hundreds of group elements to be transmitted [CFZ14]. While we do not prove the security of this variant here, the original proof by Walfish [Wal08] can be extended without issue, as this interactive DRE construction satisfies all of the required properties of the original protocol definition. As an explicit warning to implementers, we caution that the interactive ZKPK sessions must not be interleaved (i.e., the verifier must wait for all 3 message flows of the ZKPK to complete before attempting to decrypt the message). If such a pipelining approach is attempted, the deniability properties of Φ_{dre} no longer hold against online judges. We also note that, if the implementer is willing to accept the use of the random oracle model, then this instantiation of DRE can be made non-interactive through the use of the Fiat-Shamir heuristic [FS87]. However, other DRE schemes may be a better choice in such environments (see Section 3.4.4 for an incomplete survey).

3.7 An Efficient Interactive DAKE from Ring Signatures

While Φ_{dre} can be made practical through the use of interactive DRE protocols, the resulting protocol requires 9 flows to complete the key exchange. In environments with high latency, such an approach may be undesirable. Additionally, Φ_{dre} is a *non-contributory* key exchange; the resulting shared secret is chosen entirely by a single party (the responder R). As is made clear by the definition of $\mathcal{F}_{keia}^{\text{IncProc}}$, this allows an adversary that has corrupted R to cause the initiator I to use an adversarially chosen key. For some protocols, this is

not acceptable. Finally, $\mathcal{F}_{keia}^{\text{IncProc}}$ represents a *pre-specified peer* key exchange; both parties must know the identity of the other participant before the protocol begins. While Φ_{dre} begins with a flow that identifies the names of the parties (and thus the real protocol is not a pre-specified peer key exchange), the ideal functionality does not capture this notion.

All of these limitations can be overcome by a family of key exchanges known as SIGMA ("SIGn-and-MAc") protocols. First proposed by Krawczyk [Kra03], SIGMA protocols are *contributory* (both parties ensure the randomness and freshness of the resulting key), consist of only 3 message flows, and permit *post-specified peers* (i.e., the identity of the other party is an output of the protocol in addition to the key). Canetti and Krawczyk have previously shown that a basic SIGMA protocol is UC-secure in the $\mathcal{F}_{SIG}$-hybrid model with adaptive corruptions [CK02]. Unfortunately, this proof shares the limitations of all proofs in the UC model, including a failure to model public key directories that are available to the distinguishing environment. Additionally, no SIGMA protocols offer the strong deniability properties offered by Φ_{dre}.

In this section, we make use of ring signatures to construct a new deniable key exchange protocol, inspired by SIGMA designs, that offers provably strong security and deniability in the GUC framework. The resulting protocol, RSDAKE, is not a true SIGMA protocol (since it does not need to use a MAC), but it addresses all of the aforementioned problems with Φ_{dre}.

3.7.1 Ideal Functionality $\mathcal{F}_{post\text{-}keia}^{\text{IncProc}}$

Before defining RSDAKE, we begin by formulating a functionality that captures its desired properties in an ideal environment. The resulting ideal functionality, $\mathcal{F}_{post\text{-}keia}^{\text{IncProc}}$ (post-specified peer key exchange with incriminating abort), is presented in Algorithm 2. In the remainder of this section, we discuss the behavior of the functionality and the design decisions behind it.

Previously, Canetti and Krawczyk proved that the basic SIGMA protocol (upon which RSDAKE is based) is secure in a UC-based security model [CK02]. They defined an ideal functionality, $\mathcal{F}_{post\text{-}ke}$, that captures the notion of a key exchange with post-specified peers. Like $\mathcal{F}_{post\text{-}ke}$, $\mathcal{F}_{post\text{-}keia}^{\text{IncProc}}$ takes place between an unbounded number of parties, but each session captures the interaction between only two of these parties. The first party to request a key exchange is subsequently known as I, the initiator. The second party to request a key exchange is subsequently known as R, the responder. After both I and R are known, $\mathcal{F}_{post\text{-}keia}^{\text{IncProc}}$ selects a random shared key k for the session. The adversary is then given a chance to attempt to set the output (the shared key and the identity of the other party)

Algorithm 2 Ideal functionality $\mathcal{F}_{post\text{-}keia}^{\text{IncProc}}$

on receipt of $(\texttt{initiate}, sid, I, SK_I)$ **from** I:
  if (I is "active") **return**
  Mark I as "active"
  Send $(\texttt{initiate}, sid, I)$ to $\mathcal{S}$

on receipt of $(\texttt{establish}, sid, R, SK_R)$ **from** R:
  if ((I is undefined) || (I is not "active") || (R is "active") || (R is "aborted")) **return**
  Mark R as "active"
  Send $(\texttt{establish}, sid, R)$ to $\mathcal{S}$
  $k \xleftarrow{\$} \{0,1\}^\lambda$

on receipt of $(\texttt{set-key}, sid, T, P', k')$ **from** $\mathcal{S}$:
  if (a `set-key` message was already sent to T) **return**
  if (($T \notin \{I, R\}$) || (T is not "active")) **return**
  Let $P \in \{I, R\}$ such that $P \neq T$
  if (($P' \neq P$) && (P' is uncorrupted)) **return**
  if ((I is corrupt) || (R is corrupt)) {
    Send $(\texttt{set-key}, sid, P', k')$ to T
  } **else** {
    Send $(\texttt{set-key}, sid, P, k)$ to T
  }
  if (two `set-key` messages have been sent) Halt

on receipt of $(\texttt{abort}, sid, I, R)$ **from** $\mathcal{S}$:
  if (I is "active") Send delayed $(\texttt{abort}, sid, I)$ to I
  if (R is "active") {
    Mark R as "aborted"
    Send delayed $(\texttt{abort}, sid, R)$ to R
  }

on receipt of $(\texttt{incriminate}, sid, I, R)$ **from** $\mathcal{S}$:
  if (an `incriminate` message has already been received) **return**
  if ((R is "aborted") && (I is "active") && (R is honest)) {
    Execute IncProc$(sid, I, R, PK_I, PK_R, SK_R, k)$
  }

of both I and R. If the adversary has corrupted either party, then it is given the ability to send an adversarially-chosen secret k' and partner identity P to I and R.[5] Otherwise, each party is given k and the true identity of their conversation partner.

Unfortunately, $\mathcal{F}_{post\text{-}ke}$ cannot be realized in the GUC framework in the $\bar{\mathcal{G}}_{krk}^{\text{RSDAKE}}$-hybrid model. Like $\mathcal{F}_{keia}^{\text{IncProc}}$, $\mathcal{F}_{post\text{-}keia}^{\text{IncProc}}$ must explicitly weaken the deniability of the protocol by allowing for incriminating aborts. Concretely, we allow the adversary to abort the protocol in order to cause R to output incriminating information. The nature of this incriminating information is a parameter to $\mathcal{F}_{post\text{-}keia}^{\text{IncProc}}$ in the form of a procedure IncProc, allowing it to be tailored to the real protocol under consideration. When the adversary $\mathcal{S}$ asks $\mathcal{F}_{post\text{-}keia}^{\text{IncProc}}$ to abort, an instance of IncProc is started. $\mathcal{S}$ cannot cause an uncorrupted R to output a key after the protocol has aborted, but it may still cause I to output a key by withholding the `abort` message sent to I.

There is another subtle difference between $\mathcal{F}_{post\text{-}keia}^{\text{IncProc}}$ and $\mathcal{F}_{keia}^{\text{IncProc}}$. In $\mathcal{F}_{keia}^{\text{IncProc}}$, it is the initiator I that releases incriminating information when the protocol aborts. In contrast, if the RSDAKE exchange aborts, incriminating information will be released by the responder R. Since the real protocol Φ_{dre} aborts before the shared secret k is generated, $\mathcal{F}_{keia}^{\text{IncProc}}$ does not provide k as an input to IncProc. However, an incriminating abort in $\mathcal{F}_{post\text{-}keia}^{\text{IncProc}}$ occurs after R has generated k. Thus, IncProc must also accept k as input in order for the protocol to be simulatable.

3.7.2 Real Protocol RSDAKE

Our new protocol, RSDAKE, is presented in Figure 3.2. Each protocol participant P has a long-term key pair (PK_P, SK_P) for a ring signature scheme, where PK_P is publicly known. The initiator I begins a protocol session by generating an ephemeral signing key pair (pk_I, sk_I). It also generates an ephemeral Diffie-Hellman public key g^i and a ring signature key pair (rpk_I, rsk_I). It sends its identity, its ephemeral public keys, and a signature of $g^i \| rpk_I$ using pk_I to R. This signature binds the ephemeral keys for the different schemes to the same "master" key pk_I. This first message is referred to as ψ_1.

Responder R performs the same procedure and responds with pk_R, g^r, and rpk_R. It also performs a ring signature of the two ephemeral master keys pk_I and pk_R as well as the identity of I. The response message is referred to as ψ_2. The ring used for this signature

[5] $\mathcal{F}_{post\text{-}keia}^{\text{IncProc}}$ models a scenario in which the adversary $\mathcal{S}$ can completely control the value of the shared secret key after corrupting only one party. In a contributory key exchange, $\mathcal{S}$ may not have full control over this value, but it can still influence the result (e.g., by controlling the corrupted party's key contribution). Consequently, $\mathcal{F}_{post\text{-}keia}^{\text{IncProc}}$ models a more powerful adversary than is strictly needed for our purposes.

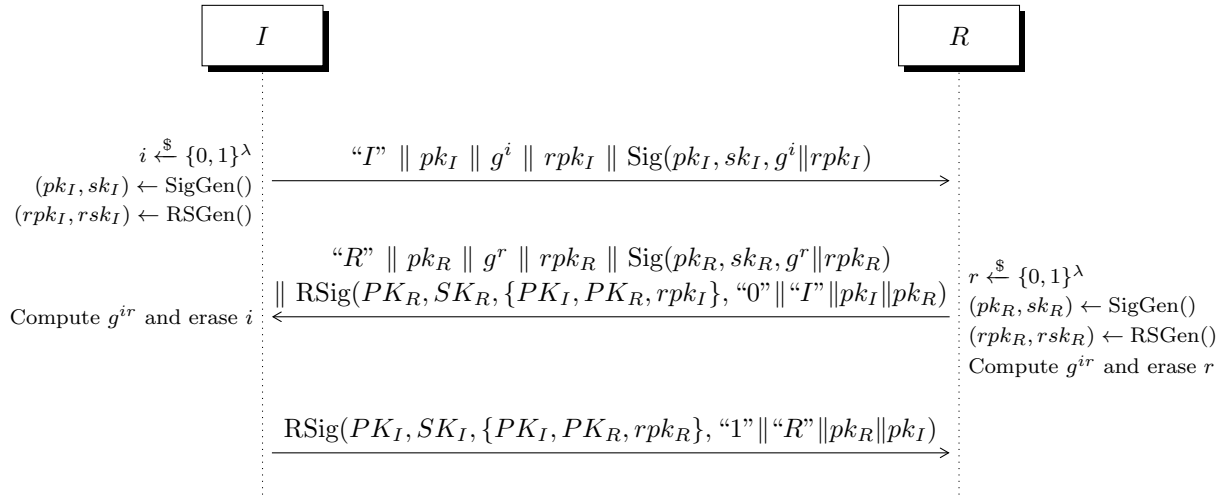


Figure 3.2: Real protocol RSDAKE. The shared secret is g^{ir}.

is $\{PK_I, PK_R, rpk_I\}$ (i.e., a ring containing the long-term keys for both parties and the ephemeral key for I). This ring signature serves the same purpose as the (traditional) signature and MAC in the basic SIGMA protocol. An honest I is convinced that R produced the signature because it knows that no other parties have access to SK_I or rsk_I. It also knows that this signature has not been reused from another session (because it contains pk_I and pk_R), and that R believes that it is communicating with the correct partner (because the signature contains the identity of I). However, this proof is not transferable to any other party because the signature could have also been forged by I using SK_I; in this way, it offers at least as much deniability as the MAC in the basic SIGMA protocol.

In the third and final step of the protocol, I responds with its own ring signature of the master ephemeral keys and the identity of R, computed over the ring $\{PK_I, PK_R, rpk_R\}$. This final message is referred to as ψ_3. R is convinced of I's identity, but cannot transfer this conviction, for the same reasons as before. The resulting shared secret is g^{ir}, as in a standard Diffie-Hellman exchange.

Unlike the basic SIGMA protocol, RSDAKE offers offline repudiation equal to that of Φ_{dre}. Specifically, anyone can forge a key exchange (and subsequent conversation) between any two parties I and R using nothing other than PK_I and PK_R. An offline forger is in the unique position of generating ephemeral keys for both simulated parties, and so it can compute both ring signatures using rsk_I and rsk_R. Transcripts generated by such a forger are indistinguishable from real transcripts due to the security of the ring signature

scheme. If the ring signature scheme provides security under full-key exposure, this indistinguishability holds even if the long-term secret keys of both I and R are subsequently compromised by the distinguisher.

3.7.3 Proof of Security

Before proving the security of RSDAKE, we must define the incriminating information that is leaked when the protocol aborts. Algorithm 3 defines IncProc for $\mathcal{F}_{post\text{-}keia}^{\text{IncProc}}$. Given this definition, we are now ready to prove the security of RSDAKE in the erasure model with fully adaptive corruptions.

Algorithm 3 IncProc$(sid, I, R, PK_I, PK_R, SK_R, k)$ for RSDAKE

on receipt of $(\texttt{incriminate}, sid, I, R, \text{“}I\text{”}, \text{“}R\text{”}, pk_I, rpk_I)$ **from** $\mathcal{S}$:

Generate $r \xleftarrow{\$} \{0,1\}^\lambda$
Generate $r_R^{SigGen} \xleftarrow{\$} \{0,1\}^\lambda$
Generate $r_R^{Sig} \xleftarrow{\$} \{0,1\}^\lambda$
Generate $r_R^{RSGen} \xleftarrow{\$} \{0,1\}^\lambda$
Generate $r_R^{RS} \xleftarrow{\$} \{0,1\}^\lambda$
Compute $(pk_R, sk_R) \leftarrow \text{SigGen}(r_R^{SigGen})$
Compute $(rpk_R, rsk_R) \leftarrow \text{RSGen}(r_R^{RSGen})$
Compute $\sigma_1 = \text{Sig}(pk_R, sk_R, g^r \| rpk_R, r_R^{Sig})$
Compute $\sigma_2 = \text{RSig}(PK_R, SK_R, \{PK_I, PK_R, rpk_I\}, \text{“0”} \| \text{“}I\text{”} \| pk_I \| pk_R, r_R^{RS})$
Compute $\psi_2 = \text{“}R\text{”} \parallel pk_R \parallel g^r \parallel rpk_R \parallel \sigma_1 \parallel \sigma_2$
Send $(\texttt{incriminate}, sid, I, R, \psi_2, r, r_R^{SigGen}, r_R^{Sig}, r_R^{RSGen}, r_R^{RS})$ to $\mathcal{S}$

Theorem 3.2 (*Security of RSDAKE*)

Assuming the existence of a signature scheme (SigGen, Sig, Vrf) and a ring signature scheme (RSGen, RSig, RVrf) that is secure under full-key exposure, RSDAKE GUC-realizes $\mathcal{F}_{post\text{-}keia}^{\text{IncProc}}$ within the erasure $\bar{\mathcal{G}}_{krk}^{\text{RSDAKE}}$-hybrid model with adaptive security when IncProc proceeds as in Algorithm 3.

Proof: To show that RSDAKE GUC-realizes $\mathcal{F}_{post\text{-}keia}^{\text{IncProc}}$, it suffices to show that RSDAKE EUC-realizes $\mathcal{F}_{post\text{-}keia}^{\text{IncProc}}$. This simplification follows from Theorem 3.1, which states that GUC-emulation is equivalent to EUC-emulation as long as RSDAKE is $\bar{\mathcal{G}}_{krk}^{\text{RSDAKE}}$-subroutine respecting.

By definition, RSDAKE EUC-realizes $\mathcal{F}_{post\text{-}keia}^{\text{IncProc}}$ if and only if, for any PPT adversary $\mathcal{A}$ attacking RSDAKE, there exists a PPT adversary $\mathcal{S}$ attacking $\mathcal{F}_{post\text{-}keia}^{\text{IncProc}}$ such that any $\bar{\mathcal{G}}_{krk}^{\text{RSDAKE}}$-externally constrained environment $\mathcal{Z}$ cannot distinguish between the real and simulated conditions.

Like most proofs in UC-based models, we will construct a simulator $\mathcal{S}$ that executes $\mathcal{A}$ internally, simulating the real protocol flows that $\mathcal{A}$ expects based on conditions in the ideal environment. To achieve the required indistinguishability property, we need to show two things: $\mathcal{Z}$ can derive no useful information from sessions other than the one under consideration, and $\mathcal{Z}$ cannot distinguish between the challenge protocols in the context of the current session. To guarantee the latter condition, we must show that, irrespective of the actions performed by $\mathcal{A}$ under the instruction of $\mathcal{Z}$, the outputs of the main parties of $\mathcal{F}_{post\text{-}keia}^{\text{IncProc}}$ are equal to those of RSDAKE, corrupted parties provide memory consistent with all other observations, and the protocol flows within the joint view of $\mathcal{A}$ and $\mathcal{Z}$ are consistent with the outputs of the main parties.

3.7.3.1 Simulator Construction

Communications between $\mathcal{A}$ and $\mathcal{Z}$: Any data sent to $\mathcal{S}$ from $\mathcal{Z}$ are copied to the input of $\mathcal{A}$. Likewise, any output from $\mathcal{A}$ is sent to $\mathcal{Z}$ by $\mathcal{S}$.

General reactions to actions by $\mathcal{A}$: If $\mathcal{A}$ sends any messages within the simulated environment that are unrelated to RSDAKE, or messages that are duplications, they are ignored (as they would be in a real network environment). If $\mathcal{A}$ delays delivery of a message flow, $\mathcal{S}$ simply waits for the flow to be delivered before continuing. This leaves $\mathcal{A}$ with few possible actions of consequence: it can alter any of the message flows it perceives, and it can corrupt simulated parties. Our model allows $\mathcal{A}$ to corrupt parties before the protocol begins, after ψ_1 has been sent, after ψ_2 has been sent, or after ψ_3 has been sent (i.e., we tolerate fully adaptive corruptions). When $\mathcal{A}$ corrupts a simulated party, $\mathcal{S}$ corrupts the corresponding ideal party in order to construct the expected state history. If $\mathcal{A}$ causes a corrupted simulated party to output a message, $\mathcal{S}$ causes the corresponding ideal party to output the same message.

Receipt of `initiate` message from $\mathcal{F}_{post\text{-}keia}^{\text{IncProc}}$: When $\mathcal{S}$ receives $(\texttt{initiate}, sid, I)$ from $\mathcal{F}_{post\text{-}keia}^{\text{IncProc}}$, it honestly constructs a ψ_1 message from the simulated party I. Specifically, it generates random coins i, r_I^{SigGen}, r_I^{Sig}, and r_I^{RSGen}, generates ephemeral key pairs $(pk_I, sk_I) \leftarrow \text{SigGen}(r_I^{SigGen})$ and $(rpk_I, rsk_I) \leftarrow \text{RSGen}(r_I^{RSGen})$, and computes

ψ_1 = "I" $\|$ pk_I $\|$ g^i $\|$ rpk_I $\|$ $\mathrm{Sig}(pk_I, sk_I, g^i \| rpk_I, r_I^{Sig})$. $\mathcal{S}$ then sends ψ_1 through $\mathcal{A}$ as if it were broadcast by the simulated party I.

Receipt of `establish` message from $\mathcal{F}_{post\text{-}keia}^{\mathbf{IncProc}}$: When $\mathcal{S}$ receives an establishment message $(\texttt{establish}, sid, R)$ from $\mathcal{F}_{post\text{-}keia}^{\mathrm{IncProc}}$, it checks to see the circumstances of the simulated ψ_1 message transmission. Since $\mathcal{F}_{post\text{-}keia}^{\mathrm{IncProc}}$ only sends an `establish` message after it has already sent an `initiate` message, ψ_1 is guaranteed to have been sent in the simulated environment (either by $\mathcal{S}$ in response to an `initiate` message or by $\mathcal{A}$ from a corrupted party).

If ψ_1 was generated by $\mathcal{S}$ and it was not modified by $\mathcal{A}$, or if simulated party I was previously corrupted, then $\mathcal{S}$ honestly constructs a ψ_2 message from the simulated party R. Specifically, it first parses ψ_1 to extract the ephemeral keys and the signature generated using sk_I. If the signature verification is successful, $\mathcal{S}$ generates random coins r, r_R^{SigGen}, r_R^{Sig}, r_R^{RSGen}, and r_R^{RS}, then generates ephemeral key pairs $(pk_R, sk_R) \leftarrow \mathrm{SigGen}(r_R^{SigGen})$ and $(rpk_R, rsk_R) \leftarrow \mathrm{RSGen}(r_R^{RSGen})$. These coins and key pairs are then used to compute two signatures: $\sigma_1 = \mathrm{Sig}(pk_R, sk_R, g^r \| rpk_R, r_R^{Sig})$ and $\sigma_2 = \mathrm{RSig}(\phi_{pk,R}, \phi_{sk,R}, \{PK_I, PK_R, rpk_I\}, \text{“0”} \| \text{“}I\text{”} \| pk_I \| pk_R, r_R^{RS})$. The key pair $(\phi_{pk,R}, \phi_{sk,R})$ used to produce σ_2 is chosen based on the prior events in the system. If $\mathcal{S}$ previously simulated the generation of ψ_1 by I, then it uses $\phi_{pk,R} = rpk_I$ and $\phi_{sk,R} = rsk_I$. If ψ_1 was sent by a corrupt simulated party I, then $\mathcal{S}$ uses its access to corrupt ideal party I to retrieve SK_I from $\bar{\mathcal{G}}_{krk}^{\mathrm{RSDAKE}}$. It then uses $\phi_{pk,R} = PK_I$ and $\phi_{sk,R} = SK_I$. $\mathcal{S}$ then constructs message ψ_2 = "R" $\|$ pk_R $\|$ g^r $\|$ rpk_R $\|$ σ_1 $\|$ σ_2.

If ψ_1' was generated by $\mathcal{S}$ but $\mathcal{A}$ altered it to ψ_1 such that $\psi_1 \neq \psi_1'$, then $\mathcal{S}$ constructs ψ_2 through the use of IncProc. $\mathcal{S}$ sends $(\texttt{abort}, sid, I, R)$ to $\mathcal{F}_{post\text{-}keia}^{\mathrm{IncProc}}$, but withholds delivery of the resulting `abort` messages to the ideal parties I and R. It then sends $(\texttt{incriminate}, sid, I, R)$ to $\mathcal{F}_{post\text{-}keia}^{\mathrm{IncProc}}$, causing an instance of IncProc to be invoked. Using the values parsed from ψ_1, $\mathcal{S}$ sends $(\texttt{incriminate}, sid, I, R, \text{“}I\text{”}, \text{“}R\text{”}, pk_I, rpk_I)$ to IncProc and receives $(\texttt{incriminate}, sid, I, R, \psi_2, r, r_R^{SigGen}, r_R^{Sig}, r_R^{RSGen}, r_R^{RS})$ in response.

$\mathcal{S}$ then sends ψ_2 through $\mathcal{A}$ as if it were sent by the simulated party R to the simulated party I.

If ψ_1 is not of the correct format, or the signature verification fails, then $\mathcal{S}$ sends $(\texttt{abort}, sid, I, R)$ to $\mathcal{F}_{post\text{-}keia}^{\mathrm{IncProc}}$ and delivers the resulting `abort` message to ideal party R immediately. The `abort` message to ideal party I is withheld by $\mathcal{S}$.

Receipt of ψ_2 by uncorrupted simulated I: When uncorrupted simulated party I receives message ψ_2 from P, it first parses ψ_2 to extract "P", pk_P, g^p, rpk_P, the signature generated using sk_P, and the ring signature. If I has previously broadcast a message ψ_1 and ψ_2 is valid, then $\mathcal{S}$ honestly constructs message ψ_3. If the signature and ring signature verify correctly, $\mathcal{S}$ generates random coins r_I^{RS}, then computes $\psi_3 = \text{RSig}(\phi_{pk,I}, \phi_{sk,I}, \{PK_I, PK_P, rpk_P\}, \text{"1"} \| \text{"}P\text{"} \| pk_P \| pk_I, r_I^{RS})$. Selection of the key pair $(\phi_{pk,I}, \phi_{sk,I})$ used to produce ψ_3 is based on the prior events in the system. If $P = R$ and $\mathcal{S}$ previously simulated the generation of ψ_2 by R, then it uses $\phi_{pk,I} = rpk_R$ and $\phi_{sk,I} = rsk_R$. If $P = R$ and ψ_2 was generated by IncProc, then $\mathcal{S}$ computes $(rpk_R, rsk_R) \leftarrow \text{RSGen}(r_R^{RSGen})$, using r_R^{RSGen} received from IncProc, and uses $\phi_{pk,I} = rpk_R$ and $\phi_{sk,I} = rsk_R$. If ψ_2 was sent by a corrupt simulated party P, then $\mathcal{S}$ uses its access to corrupt ideal party P to retrieve SK_P from $\bar{\mathcal{G}}_{krk}^{\text{RSDAKE}}$. It then uses $\phi_{pk,I} = PK_P$ and $\phi_{sk,I} = SK_P$. $\mathcal{S}$ then sends ψ_3 through $\mathcal{A}$ as if it were sent by the simulated party I to the simulated party P.

In addition to sending ψ_3, $\mathcal{S}$ also causes ideal party I to output a key. It computes $k' = g^{ip} = (g^p)^i$, where i is the secret key used to generate ψ_1, and sends $(\texttt{set-key}, sid, I, P, k')$ to $\mathcal{F}_{post\text{-}keia}^{\text{IncProc}}$. The resulting `set-key` message to I is delivered immediately.

If I has not previously broadcast a message ψ_1, then the message is ignored. If ψ_2 is not of the correct form, or either signature verification fails, then $\mathcal{S}$ sends $(\texttt{abort}, sid, I, R)$ to $\mathcal{F}_{post\text{-}keia}^{\text{IncProc}}$ and delivers the resulting `abort` message to ideal party I immediately. The `abort` message to ideal party R is withheld by $\mathcal{S}$.

Receipt of ψ_3 by uncorrupted simulated R: When uncorrupted simulated party R receives message ψ_3 from I, it first checks to ensure that it has previously received a message ψ_1 from I and that it sent a response ψ_2. If either of these conditions do not hold, then the message is ignored. ψ_3 is then interpreted as a ring signature and verified. If the ring signature is invalid or fails to verify, then $\mathcal{S}$ sends $(\texttt{abort}, sid, I, R)$ to $\mathcal{F}_{post\text{-}keia}^{\text{IncProc}}$ and delivers the resulting `abort` message to ideal party R immediately. The `abort` message to ideal party I is withheld by $\mathcal{S}$.

If the ring signature is valid, then $\mathcal{S}$ also causes ideal party R to output a key. It computes $k' = g^{ir} = (g^i)^r$, where r is the secret key used to generate ψ_2, and sends $(\texttt{set-key}, sid, R, I, k')$ to $\mathcal{F}_{post\text{-}keia}^{\text{IncProc}}$. The resulting `set-key` message to R is delivered immediately.

Transmission of ψ_1 by corrupted simulated I: When $\mathcal{S}$ has not yet received an `initiate` message from $\mathcal{F}_{post\text{-}keia}^{\text{IncProc}}$, but $\mathcal{A}$ causes a corrupted simulated party I to issue message ψ_1, then $\mathcal{S}$ must reflect this in the ideal environment. $\mathcal{S}$ uses its corruption of the corresponding ideal party I to retrieve SK_I from $\bar{\mathcal{G}}_{krk}^{\text{RSDAKE}}$. It then sends $(\texttt{initiate}, sid, I, SK_I)$ to $\mathcal{F}_{post\text{-}keia}^{\text{IncProc}}$, but ignores the resulting `initiate` message sent by $\mathcal{F}_{post\text{-}keia}^{\text{IncProc}}$.

Transmission of ψ_2 by corrupted simulated R: When $\mathcal{S}$ has not yet received an `establish` message from $\mathcal{F}_{post\text{-}keia}^{\text{IncProc}}$, but $\mathcal{A}$ causes a corrupted simulated party R to issue message ψ_2, then $\mathcal{S}$ must reflect this in the ideal environment. $\mathcal{S}$ uses its corruption of the corresponding ideal party R to retrieve SK_R from $\bar{\mathcal{G}}_{krk}^{\text{RSDAKE}}$. It then sends $(\texttt{establish}, sid, R, SK_R)$ to $\mathcal{F}_{post\text{-}keia}^{\text{IncProc}}$, but ignores the resulting `establish` message sent by $\mathcal{F}_{post\text{-}keia}^{\text{IncProc}}$.

Constructing state for corrupted parties: When $\mathcal{A}$ corrupts a party in the simulated environment, $\mathcal{S}$ corrupts the corresponding party in the ideal environment. In addition, $\mathcal{S}$ must provide $\mathcal{A}$ with a simulated historical state for the corrupted party.

If $\mathcal{A}$ corrupts the party known as I after an `initiate` message has been received, then $\mathcal{S}$ provides the random coins r_I^{SigGen}, r_I^{Sig}, and r_I^{RSGen} used to construct ψ_1. If I has also sent ψ_3 at the time of the corruption, the random coins r_I^{RS} are provided. If the corruption occurs after ψ_1 was sent but before ψ_2 was received by simulated party I, then $\mathcal{S}$ also provides the secret exponent i. If ψ_1 was sent but ψ_2 was already received, then $\mathcal{S}$ provides the session key k generated by $\mathcal{F}_{post\text{-}keia}^{\text{IncProc}}$ and retrieved during the corruption of the ideal party I.

If $\mathcal{A}$ corrupts the party known as R after an `establish` message has been received, then $\mathcal{S}$ provides the random coins r_R^{SigGen}, r_R^{Sig}, and r_R^{RSGen} used to construct ψ_2. $\mathcal{S}$ also provides the session key k generated by $\mathcal{F}_{post\text{-}keia}^{\text{IncProc}}$ and retrieved during the corruption of the ideal party R.

3.7.3.2 Proof of Indistinguishability

We now turn to the task of proving that $\mathcal{S}$ acting on $\mathcal{F}_{post\text{-}keia}^{\text{IncProc}}$ is indistinguishable from $\mathcal{A}$ acting on RSDAKE. To do this, we divide all possible behaviors of $\mathcal{A}$ into several cases. For each case, we show that the protocol flows generated by $\mathcal{S}$ are indistinguishable from those generated by RSDAKE, outputs from $\mathcal{F}_{post\text{-}keia}^{\text{IncProc}}$ are indistinguishable from

those from RSDAKE, and that the simulated memory states of corrupted parties are indistinguishable from those of real parties.

The honest case: This situation occurs when $\mathcal{A}$ does not corrupt I or R until after the session concludes, or alter any message flows.

All three messages are generated by $\mathcal{S}$ honestly (i.e., exactly how they would be generated by the parties in a real RSDAKE session), with the exception of the ring signatures. The ring signatures are not signed by the long-term secret keys of the parties, as in a real interaction. Instead, they are signed by the ephemeral key of the opposite party. However, if $\mathcal{Z}$ was able to distinguish the signatures produced by $\mathcal{S}$ from those produced in a real interaction, then $\mathcal{Z}$ would be able to break the security of the underlying ring signature scheme. Since we make use of a scheme that is secure under full-key exposure, this is true even if I and R are corrupted after the session concludes (and thus all secret keys in the ring are within the joint view of $\mathcal{A}$ and $\mathcal{Z}$). Since this is a contradiction, the message flows are indistinguishable from real flows.

Since neither party is corrupted, the output from I and R in the ideal environment includes the correct identity of the conversation partner, as well as the shared secret k randomly generated by $\mathcal{F}_{post\text{-}keia}^{\text{IncProc}}$. These are the expected party identities from the real interaction, so the only possible way for $\mathcal{Z}$ to distinguish between real and simulated outputs is by examining k. Since i and r are erased by real parties before they return output, $\mathcal{A}$ cannot access these values, even when corrupting simulated I and R after the session concludes. Therefore, any ability to distinguish between challenge protocols based on the choice of k would mean that $\mathcal{Z}$ could distinguish between k and g^{ir}. This is only possible if $\mathcal{Z}$ can break the decisional Diffie-Hellman assumption within the group containing g, which we assume is not possible.

Finally, corruption of simulated I or R produces memory states containing only random coins used for generation of the messages. These coins are produced by $\mathcal{S}$ using the same technique as honest parties, and thus these memory states are indistinguishable from real ones.

Alteration of ψ_1: This situation occurs when ψ_1 generated by $\mathcal{S}$ is altered by $\mathcal{A}$ in transit, but neither I nor R are corrupted when ψ_1 is delivered.

When ψ_1 is altered, $\mathcal{S}$ generates ψ_2 from R using IncProc. The definition of IncProc involves honestly generating ψ_2 using the long-term secret key of R, so this flow is indistinguishable from a real message from R. Likewise, the memory state of R is

indistinguishable from the real situation because IncProc provides $\mathcal{S}$ with the random coins used to generate the ephemeral keys in ψ_2.

$\mathcal{S}$ causes the protocol to abort, but does not deliver abort messages to either party. If $\mathcal{A}$ allows ψ_2 to be delivered to I, then I will abort. This matches the output of real interactions because I expects ψ_2 to include a ring signature over the true pk_I generated by I. The only way for the simulated and real situations to differ is if $\mathcal{A}$ somehow alters ψ_2 so that it is a valid response. Since $\mathcal{A}$ does not possess any of the secret keys in the expected ring, this would violate the security properties of the underlying ring signature scheme.

Alteration of ψ_2: This situation occurs when ψ_2 generated by $\mathcal{S}$ is altered by $\mathcal{A}$ in transit, but neither I nor R are corrupted when ψ_2 is delivered. $\mathcal{S}$ causes I to immediately abort when it receives an altered ψ_2. As mentioned previously, I will always abort because the ring signature in the altered ψ_2 message cannot be correct.

Indistinguishability under corruptions: This situation occurs when either party is corrupted at a time before the times covered by the previous cases.

The only difference between the normal operation of $\mathcal{S}$ and this case is the secret key used to compute the ring signatures in messages generated by $\mathcal{S}$. Whereas $\mathcal{S}$ normally uses the ephemeral signing keys rsk_I and rsk_R to sign the ring signature produced by the other party, these keys might not be generated by $\mathcal{S}$ when a party is corrupted before sending its first message. However, $\mathcal{S}$ instead makes use of the long-term secret key of the corrupted party to sign the ring signatures. Again, these message flows are indistinguishable from real flows due to the security of the ring signature scheme. The outputs of the protocol are indistinguishable because the uncorrupted party is simulated honestly.

If both simulated parties are corrupted, then indistinguishability is trivial. $\mathcal{S}$ never generates any messages, and so they cannot be used by $\mathcal{Z}$ to detect simulation. The outputs of corrupted parties are copied to the outputs of the corresponding ideal parties, so this is also not useful to $\mathcal{Z}$.

In all cases of corruption, $\mathcal{S}$ provides the expected memory state for the corrupted party—the set of random coins used to generate ephemeral signing keys, and possibly some secret keys (depending on which party is corrupted and when). In all cases, these values are indistinguishable from real values because the parties are simulated honestly.

Data from other sessions: Since we are considering the security of RSDAKE in the EUC model, we must also consider the usefulness of information collected by $\mathcal{Z}$ from other protocol sessions. No information from other sessions can be used to assist $\mathcal{A}$ with the generation of false message flows: ψ_1 is generated using no long-term information, and both ψ_2 and ψ_3 require computation of a ring signature bound to the contents of ψ_1. Due to the security of the ring signature scheme, no external information is useful when generating valid message flows. ■

The security of RSDAKE relies on several assumptions: the hardness of the DDH problem in the group generated by g, the security of the signature scheme, and the security of the ring signature scheme. The exact set of security assumptions is defined by the choice of these underlying schemes. In Chapter 4, we consider a particular instantiation of RSDAKE and list the complete set of resulting assumptions.

3.8 A Non-Interactive Deniable Key Exchange

Both Φ_{dre} and RSDAKE have a usability limitation: they are interactive protocols. Both parties must be online to complete the key exchange. In applications such as secure messaging, the key exchange must be completed before messages can be transmitted. In some domains, such as instant messaging, consistent peer availability may be a valid assumption. However, email and text messaging are two extremely popular systems in which interactive key exchanges cannot be used in general. These environments benefit from the use of non-interactive key exchanges; secure messages can be sent immediately to any peer in the network, irrespective of their current connectivity.

In this section, we present a secure and deniable one-round key exchange protocol that can be used in interactive or non-interactive settings. Specifically, we would like a protocol that can be used as the initial key exchange for TextSecure, one of the most promising secure messaging schemes identified in Chapter 2. We begin by formalizing the notion of a non-interactive deniable key exchange by defining a new ideal functionality in the GUC framework. Next, we present our new protocol, Spawn*, and prove its security in interactive environments. Unfortunately, it is not possible to maintain all desired security properties in non-interactive settings. We discuss the consequences of using Spawn* non-interactively, and show that it is nonetheless an improvement over existing approaches. Next, we demonstrate how a relaxation of the security model admits a variant, Spawn, with significantly improved performance in practice. Finally, we explain how Spawn* (or Spawn) can be used as a bootstrap for the Axolotl key ratchet, permitting it to be used in the TextSecure protocol.

3.8.1 Ideal Functionality $\mathcal{F}_{1psp\text{-}keia}^{\mathbf{IncProc}}$

To prove that our key exchange is secure in the GUC model, we must define an ideal protocol that captures the functionality of 3-DH with prekeys (the scheme used by TextSecure). Unfortunately, neither $\mathcal{F}_{keia}^{\text{IncProc}}$ nor $\mathcal{F}_{post\text{-}keia}^{\text{IncProc}}$ fully describe the desired properties.

In TextSecure, the initiator I begins by uploading ephemeral prekeys to a central server. Subsequently, the responder R requests the next available prekey for I and uses it to complete the key exchange. In practice, the first message of the conversation is encrypted under this key and attached to the same flow. Even if I is offline when this message is sent, and R then goes offline forever, I will still be able to decrypt this message when it comes back online. It is important to note that I does not know the identity of the party that will respond to the prekeys it produces, but R knows the identity of the party to whom it wishes to send a message. In this sense, the key exchange has a *single post-specified peer*. Concretely, the identity of R should be part of the *output* for I, while the identity of I is an *input* for R. Neither $\mathcal{F}_{keia}^{\text{IncProc}}$ nor $\mathcal{F}_{post\text{-}keia}^{\text{IncProc}}$ captures this notion.

Similarly to $\mathcal{F}_{post\text{-}keia}^{\text{IncProc}}$, information generated by IncProc incriminates the responder R in the TextSecure setting (rather than the initiator I as in $\mathcal{F}_{keia}^{\text{IncProc}}$). Additionally, IncProc is called after the shared secret k is generated. Therefore, IncProc must also accept k as input in order for the protocol to be simulatable.

With these differences in mind, we define ideal functionality $\mathcal{F}_{1psp\text{-}keia}^{\text{IncProc}}$ (single post-specified peer key exchange with incriminating abort) to capture the desired protocol properties. $\mathcal{F}_{1psp\text{-}keia}^{\text{IncProc}}$ takes place between two or more parties. The functionality is given in Algorithm 4.

In the normal case, a party informs $\mathcal{F}_{1psp\text{-}keia}^{\text{IncProc}}$ that it would like to solicit a connection. Without loss of generality, we call this party I. Only one party solicits a connection per session. Later, another party asks the functionality to complete an exchange with I. Without loss of generality, we call this party R. Only one party responds to a solicitation per session. $\mathcal{F}_{1psp\text{-}keia}^{\text{IncProc}}$ generates a shared secret k, and then waits for the adversary $\mathcal{S}$ to issue a `set-key` request. If $\mathcal{S}$ has not corrupted R, then $\mathcal{F}_{1psp\text{-}keia}^{\text{IncProc}}$ instantly sends k to R and then sends k and the identity of R as a delayed message to I.

$\mathcal{S}$ can choose to withhold the final message sent to I, causing R to complete and I to stall. If $\mathcal{S}$ corrupts R before the protocol halts, then it can cause I to receive an adversarially-chosen secret k' and the identity of any corrupted party P (i.e., $\mathcal{S}$ can effectively instruct R to identify itself as any other party under the control of $\mathcal{S}$). Finally, $\mathcal{S}$ is also allowed to abort the protocol after I has solicited a connection, and it can choose whether or not I should be informed of this abort (by choosing to withhold or allow the

Algorithm 4 Ideal functionality $\mathcal{F}_{1psp\text{-}keia}^{\text{IncProc}}$

on receipt of $(\texttt{solicit}, sid, I, SK_I)$ **from** I:
  if ((I is "active") || (I is "aborted")) **return**
  Mark I as "active"; record $(\texttt{initiator}, sid, I, SK_I)$
  Send $(\texttt{solicit}, sid, I)$ to $\mathcal{S}$

on receipt of $(\texttt{establish}, sid, I, R, SK_R)$ **from** R:
  if (($\texttt{solicit}$ not received) || (R is "active")) **return**
  Mark R as "active"; record $(\texttt{responder}, sid, R, SK_R)$
  Send $(\texttt{establish}, sid, I, R)$ to $\mathcal{S}$
  $k \xleftarrow{\$} \{0,1\}^\lambda$

on receipt of $(\texttt{set-key}, sid, P, k')$ **from** $\mathcal{S}$:
  if ((k is set) && ($P = R$ || P is corrupt)) {
    if (R is corrupt) {
      Send $(\texttt{set-key}, sid, I, P, k')$ to R
      if (I is "active") {
        Send delayed $(\texttt{set-key}, sid, I, P, k')$ to I
      }
    } **else** {
      Send $(\texttt{set-key}, sid, I, R, k)$ to R
      if (I is "active") {
        Send delayed $(\texttt{set-key}, sid, I, R, k)$ to I
      }
    }
  }
  Halt

on receipt of $(\texttt{abort}, sid, I, R)$ **from** $\mathcal{S}$:
  if (I is "active") {
    Mark I as "aborted"
    Send delayed $(\texttt{abort}, sid, I, R)$ to I
  }

on receipt of $(\texttt{incriminate}, sid, R)$ **from** $\mathcal{S}$:
  if (already received $\texttt{incriminate}$ message) **return**
  if ((I is "aborted") && (R is "active") && (R is honest)) {
    Execute IncProc$(sid, I, R, PK_I, PK_R, SK_R, k)$
  }

`abort` message sent to I). In any case, if the protocol is aborted, $\mathcal{S}$ can cause R to generate incriminating information that proves R was attempting to communicate with I. $\mathcal{S}$ cannot cause I to output a key if the protocol was aborted, but it can still cause R to output a key.

$\mathcal{F}_{1psp\text{-}keia}^{\text{IncProc}}$ is parameterized with a procedure IncProc that accepts the as parameters the session identifier, the identities of I and R, the long-term public keys of I and R, the secret key of R, and the shared secret k. Similarly to $\mathcal{F}_{keia}^{\text{IncProc}}$, this IncProc is meant to capture the non-simulatable incriminating information sent to the adversary when the functionality is aborted.

3.8.2 Real Protocol Spawn*

We now define a one-round DAKE, Spawn*, that can be used in interactive or non-interactive settings. Spawn* provides both offline and online message and participation repudiation (with limited exceptions that we will discuss later) while also providing forward secrecy.[6] The protocol, which takes place between an initiator I and a responder R, is depicted in Figure 3.3.

Before any sessions begin, all parties register long-term public keys with a PKI. Each party P generates a master keypair $(PK_P^{Sig}, SK_P^{Sig}) \leftarrow \text{SigGen}()$, and two scheme-specific keypairs $(PK_P^{DRE}, SK_P^{DRE}) \leftarrow \text{DRGen}()$ and $(PK_P^{RS}, SK_P^{RS}) \leftarrow \text{RSGen}()$. P computes a signature $\sigma_P^{PKI} \leftarrow \text{Sig}(PK_P^{Sig}, SK_P^{Sig}, PK_P^{DRE} \| PK_P^{RS})$ to bind the keys together, and then uploads $PK_P = (PK_P^{Sig}, PK_P^{DRE}, PK_P^{RS}, \sigma_P^{PKI})$ to the PKI along with proofs of knowledge of the corresponding secret keys. When retrieving PK_P from the PKI, parties verify the trustworthiness of PK_P^{Sig} using a trust establishment scheme. This trust is extended to PK_P^{DRE} and PK_P^{RS} by verifying σ_P^{PKI}.

When starting a session, I generates an ephemeral key pair (pk, sk) for a standard signature scheme. This key pair is then used to sign ephemeral public keys for other schemes: one pair for non-committing encryption, and one pair for ring signatures. This signature binds both ephemeral keys to pk. In an interactive setting, I immediately sends its identity, the ephemeral public keys, and the signature binding them to pk, to R as message ψ_1. In a non-interactive setting, I instead uploads this information as a prekey

[6] In a well-known result, Bellare et al. have previously shown that no one-round protocol can achieve the strongest notion of forward secrecy [BPR00]. In this section, we consider a weaker form of forward secrecy: an adversary cannot recover the shared secret key from an unmanipulated protocol session (i.e., a session in which no flows were altered) by later compromising any long-term secret keys. This notion is equivalent to the "weak forward secrecy" of Bellare et al.

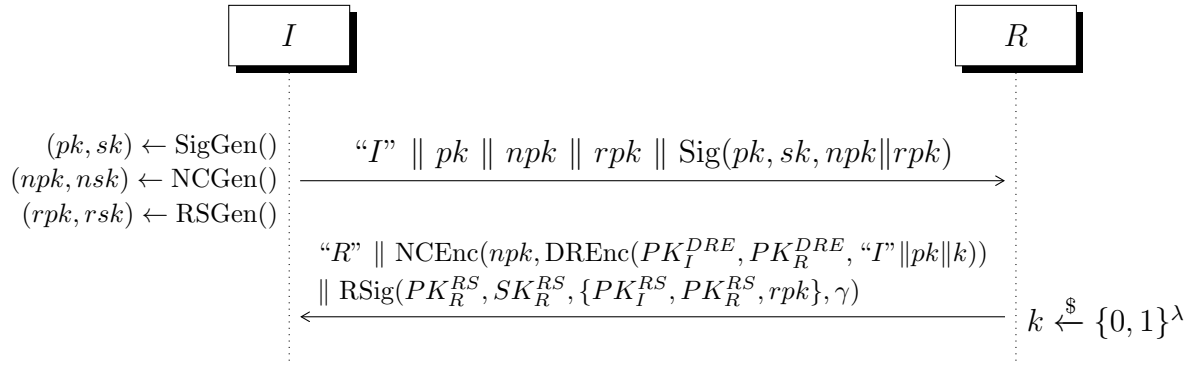


Figure 3.3: Real protocol Spawn*. The shared secret is k. γ refers to "R" concatenated with the ciphertext produced by NCEnc.

to a central server. Later, when R wishes to send a message to I, the central server relays the keys to R. Note that I expects a single response using each prekey. Thus, the role of the central server is merely to prevent prekey collisions (i.e., if multiple responders were to use the same prekey to send messages to I). For this reason, the central server is not trusted—while it can attack the availability of the protocol by refusing to relay keys, or by distributing non-genuine or repeated prekeys, it is not entrusted with any secret information that could be used to attack the security of the protocol (e.g., the central server cannot cause message exposure by altering the prekey). In effect, the central server in the non-interactive setting is strictly weaker than an active network attacker in the interactive setting.

Irrespective of the interactivity mode, R subsequently verifies the consistency of ψ_1 and then uses the ephemeral keys to formulate a response ψ_2 to I. At a high level, this response consists of R's identity, a wrapped session key k, and a signature to authenticate the ciphertext. The details of the encryption and ring signature are somewhat unintuitive. The session key is first encrypted using dual-receiver encryption for the long-term public keys of I and R. The resulting ciphertext is then encrypted once more using non-committing encryption for npk. Consequently, the session key can only be decrypted by a party knowing $\{nsk, SK_I^{DRE}\}$ or $\{nsk, SK_R^{DRE}\}$. In the normal case (when both I and R are honest), the only party satisfying these requirements is I (since R does not know nsk). The reasons for these choices will become clear when we prove the security of the protocol.

R also encrypts pk and the identifier for I alongside k. This binds both the non-committing encryption and the dual-receiver encryption to the current session, preventing replay attacks.

As part of the response ψ_2, R also includes a ring signature of the ciphertext. The ring, $\{PK_I^{RS}, PK_R^{RS}, rpk\}$, includes the long-term public keys of both I and R, as well as the ephemeral key rpk associated with pk. Like the choice of keys for the encryption scheme, the reasons for this choice will become clear in the security proof. In the case where both I and R are honest, R creates the signature using its long-term secret SK_R^{RS}. An honest I is assured that only R could have produced the signature, because I knows that it has not revealed SK_I^{RS} or rsk to any other party.

3.8.3 Unrigging Non-Committing Encryption

Our proof of security of Spawn* makes use of a slightly unintuitive quirk of non-committing encryption. We will frequently make use of NCSim to "rig" a public key pk and ciphertext γ with auxiliary information α. However, in some cases γ will be ignored and a ciphertext γ' will be produced using $\gamma' \leftarrow \text{NCEnc}(pk, m)$ for some message m. In these cases, we may need to "unrig" pk by producing a corresponding secret key sk that can decrypt γ'.

To accomplish this task, we can simply perform $(sk, r^*, r^{NCE}) \leftarrow \text{NCEqv}(pk, \gamma, \alpha, \phi)$, where ϕ is any arbitrary message. Now we have "unrigged" pk to produce a key pair (pk, sk) that can be used normally. It is necessarily true that $\text{NCDec}(pk, sk, \gamma') = m$ (i.e., we can retroactively decrypt γ' by unrigging pk). This property follows from the fact that $(pk, sk) \leftarrow \text{NCGen}(r^*)$ is a valid key generation call that satisfies the correctness properties of non-committing encryption (i.e., it can be used to encrypt and decrypt messages as in a traditional asymmetric cryptosystem).

3.8.4 Proof of Interactive Spawn* Security

Before proving the security of Spawn*, we must define the incriminating information that is leaked when the protocol aborts. Algorithm 5 defines IncProc for $\mathcal{F}_{1psp\text{-}keia}^{\text{IncProc}}$.

Note that long-term keypairs in Spawn* actually consist of three keypairs—each for use with a different cryptosystem. While it is possible to redefine $\mathcal{F}_{1psp\text{-}keia}^{\text{IncProc}}$ to explicitly handle these individual keys, it is preferable to define ideal functionalities as generically as possible to encourage reuse. Instead, we assume that the PKI, represented by the ideal functionality $\bar{\mathcal{G}}_{krk}$, stores $PK_P = (PK_P^{Sig}, PK_P^{DRE}, PK_P^{RS}, \sigma_P^{PKI})$ and $SK_P = (SK_P^{Sig}, SK_P^{DRE}, SK_P^{RS})$ for each party P.

Although I will typically initially know the identity of R in an interactive setting, we will prove that the protocol is secure even when this is not the case (i.e., I broadcasts

Algorithm 5 IncProc$(sid, I, R, PK_I, PK_R, SK_R, k)$ for Spawn*

on receipt of $(\texttt{incriminate}, sid, I, R, pk, npk, rpk)$ **from** $\mathcal{S}$:

Parse $PK_I = (PK_I^{Sig}, PK_I^{DRE}, PK_I^{RS}, \sigma_I^{PKI})$

Parse $PK_R = (PK_R^{Sig}, PK_R^{DRE}, PK_R^{RS}, \sigma_R^{PKI})$

Parse $SK_R = (SK_R^{Sig}, SK_R^{DRE}, SK_R^{RS})$

Generate $k \xleftarrow{\$} \{0,1\}^\lambda$

Generate $r^{DRE} \xleftarrow{\$} \{0,1\}^\lambda$

Generate $r^{NCE} \xleftarrow{\$} \{0,1\}^\lambda$

Generate $r^{RS} \xleftarrow{\$} \{0,1\}^\lambda$

Compute $\gamma = \text{NCEnc}(npk, \text{DREnc}(PK_I^{DRE}, PK_R^{DRE}, \text{“}I\text{”}\|pk\|k, r^{DRE}), r^{NCE})$

Compute $\sigma = \text{RSig}(PK_R^{RS}, SK_R^{RS}, \{PK_I^{RS}, PK_R^{RS}, rpk\}, \gamma, r^{RS})$

Compute $\psi_2 = \text{“}R\text{”}\|\gamma\|\sigma$

Send $(\texttt{incriminate}, sid, I, R, \psi_2, r^{DRE}, r^{NCE}, r^{RS})$ to $\mathcal{S}$

ψ_1 to all parties without knowing which will respond). One motivating factor behind this decision is to easily facilitate extension of the proof to the non-interactive setting. We are now prepared to prove that Spawn* is a secure deniable implementation of the ideal functionality $\mathcal{F}_{1psp\text{-}keia}^{\text{IncProc}}$, even when we disallow erasures and allow fully adaptive corruptions.

Theorem 3.3 (*Security of Spawn**)

Assuming the existence of a signature scheme (SigGen, Sig, Vrf), ring signature scheme (RSGen, RSig, RVrf) that is secure under full-key exposure, a dual-receiver encryption scheme (DRGen, DREnc, DRDec), and a non-committing public-key cryptosystem (NCGen, NCEnc, NCDec, NCSim, NCEqv), Spawn* GUC-realizes $\mathcal{F}_{1psp\text{-}keia}^{\text{IncProc}}$ within the non-erasure $\bar{\mathcal{G}}_{krk}^{\text{Spawn}^*}$-hybrid model with adaptive security when IncProc proceeds as in Algorithm 5.

Proof: To show that Spawn* GUC-realizes $\mathcal{F}_{1psp\text{-}keia}^{\text{IncProc}}$, we show that Spawn* EUC-realizes $\mathcal{F}_{1psp\text{-}keia}^{\text{IncProc}}$. Our proof of this is similar to the proof of Theorem 3.2. We construct a simulator $\mathcal{S}$ that simulates an execution of Spawn* for the real adversary $\mathcal{A}$, while relaying messages between $\mathcal{Z}$ and $\mathcal{A}$. We need to show two things: $\mathcal{Z}$ can derive no useful information from sessions other than the one under consideration, and $\mathcal{Z}$ cannot distinguish between the challenge protocols in the context of the current session. To guarantee the latter condition, we must show that, irrespective of the actions performed

$\mathcal{A}$ Actions				$\mathcal{S}$ Actions				
Corruptions		**Alterations**		**Generation**		**Outputs**		
I	R	ψ_1	ψ_2	ψ_1	ψ_2	I	R	**Case**
C	C	No	No	NCSim	NCEqv	k, R	k	3.8.4.1
C	C	No	Yes	NCSim	NCEqv	Abort	k	3.8.4.2
C	C	Yes	Any	NCSim	IncProc	Abort	k	3.8.4.3
A	C	Any	Any	$\mathcal{A}$ Picks	I Signs	Corrupt	k	3.8.4.4
C	A/B	Any	Any	NCSim	$\mathcal{A}$ Picks	k', P	Corrupt	3.8.4.5
B	C	Any	Any	NCSim	I Signs	Corrupt	k	3.8.4.6
A/B	A/B	Any	Any	NCSim	$\mathcal{A}$ Picks	Corrupt	Corrupt	3.8.4.7

Table 3.1: Behavior of $\mathcal{S}$ when simulating $\mathcal{A}$. For corruptions, "A" refers to corruption before ψ_1 is sent, "B" to corruption after ψ_1 is sent but before ψ_2 is sent, and "C" to corruption after ψ_2 is sent (or no corruption at all).

by $\mathcal{A}$ under the instruction of $\mathcal{Z}$, the outputs of the main parties of $\mathcal{F}_{1psp\text{-}keia}^{\text{IncProc}}$ are equal to those of Spawn*, corrupted parties provide memory consistent with all other observations, and the protocol flows observed by $\mathcal{A}$ (and available to $\mathcal{Z}$) are consistent with the outputs of the main parties.

$\mathcal{S}$ will need to behave differently based on corruptions and message alterations performed by $\mathcal{A}$. There are seven possible behaviors of $\mathcal{S}$ that cover all possible behaviors of $\mathcal{A}$. In each case, $\mathcal{S}$ chooses a mechanism for generating ψ_1 and ψ_2, and an approach for inducing outputs from the ideal parties I and R. Table 3.1 provides an overview of the actions performed by $\mathcal{S}$ based on the actions of $\mathcal{A}$. Next, we describe the actions of $\mathcal{S}$ in every possible case.

3.8.4.1 Normal Operation

This situation occurs when $\mathcal{A}$ does not corrupt either party or alter any message flows. In other words, this is the "normal" situation in which both I and R output success. However, for the purposes of the proof, we never consider a case where a party is never corrupted—we always assume that a party is corrupted after it returns output to $\mathcal{Z}$ (or earlier). The reason for this is that $\mathcal{Z}$ can always potentially learn more by corrupting a party after protocol completion, but never less; choosing to never corrupt a party is equivalent to corrupting the party after the protocol and ignoring the revealed state. Thus, in this case we consider the behavior of $\mathcal{S}$ when $\mathcal{A}$ does nothing except corrupting I and R after ψ_2 is received by I.

$\mathcal{S}$ waits until it receives a $(\texttt{solicit}, sid, I)$ message from $\mathcal{F}_{1psp\text{-}keia}^{\text{IncProc}}$. It then computes $(pk, sk) \leftarrow \text{SigGen}()$, $(npk, \gamma, \alpha) \leftarrow \text{NCSim}()$, and $(rpk, rsk) \leftarrow \text{RSGen}()$. ψ_1 is then constructed as $\psi_1 = \text{“}I\text{”}\|pk\|npk\|rpk\|\text{Sig}(pk, sk, npk\|rpk)$. $\mathcal{S}$ sends ψ_1 through $\mathcal{A}$, which does not alter it.

$\mathcal{S}$ waits until it receives an $(\texttt{establish}, sid, I, R)$ message from $\mathcal{F}_{1psp\text{-}keia}^{\text{IncProc}}$. At this point, $\mathcal{S}$ generates random coins $r^{RS} \xleftarrow{\$} \{0,1\}^\lambda$ and uses them to compute the signature $\sigma = \text{RSig}(rpk, rsk, \{PK_I^{RS}, PK_R^{RS}, rpk\}, \gamma, r^{RS})$. It then sends $\psi_2 = \gamma\|\sigma$ through $\mathcal{A}$, which does not alter it.

$\mathcal{S}$ now sends $(\texttt{set-key}, sid, R, 0)$ to $\mathcal{F}_{1psp\text{-}keia}^{\text{IncProc}}$, causing a session key k to be sent to ideal parties I and R (although this key is hidden from $\mathcal{S}$). $\mathcal{S}$ does not delay delivery of these `set-key` messages. Ideal party I will output (k, R), and ideal party R will output k—exactly as expected from a real session.

When $\mathcal{A}$ corrupts either party, the session key k generated by $\mathcal{F}_{1psp\text{-}keia}^{\text{IncProc}}$ is revealed to $\mathcal{S}$ (since it corrupts the corresponding ideal party and retrieves its message history). Now $\mathcal{S}$ can open the non-commiting encryption such that it appears to show the expected encryption of k. It begins by generating random coins $r^{DRE} \xleftarrow{\$} \{0,1\}^\lambda$. Next, $\mathcal{S}$ uses $(nsk, r^*, r^{NCE}) \leftarrow \text{NCEqv}(npk, \gamma, \alpha, \text{DREnc}(PK_I^{DRE}, PK_R^{DRE}, \text{“}I\text{”}\|pk\|k, r^{DRE}))$ to construct the historical state of the simulated parties I and R. Specifically, from the perspective of $\mathcal{A}$, the history of I appears to show $(npk, nsk) \leftarrow \text{NCGen}(r^*)$, and the history of R appears to show that ψ_2 contained a valid encryption of k using random coins r^{DRE} and r^{NCE}. By the properties of non-committing encryption, even though the two flows ψ_1 and ψ_2 are different from the flows that would be generated by honest parties, they are indistinguishable from real flows from the perspective of $\mathcal{A}$. Additionally, the security of the ring signature scheme with respect to full key exposure ensures that the ring signature signed by rsk cannot be distinguished from a real one signed by SK_R^{RS}, even when both parties are corrupted.

3.8.4.2 Alteration of ψ_2

This situation occurs when $\mathcal{A}$ does not corrupt either party until the protocol completes and does not modify ψ_1, but modifies ψ_2. In this case, $\mathcal{S}$ generates ψ_1 using NCSim and ψ_2 using NCEqv as in the normal case (Case 3.8.4.1). However, once $\mathcal{S}$ notices that $\mathcal{A}$ alters ψ_2, it must abort the protocol. $\mathcal{S}$ sends $(\texttt{abort}, sid, I, R)$ to $\mathcal{F}_{1psp\text{-}keia}^{\text{IncProc}}$, and does not delay delivery of the abort message to I. Ideal party I will output that the protocol was aborted, and R will output k, as expected for the real protocol. If $\mathcal{A}$

subsequently corrupts either party, the memory of the corrupted party is reconstructed using NCEqv as in Case 3.8.4.1.

Causing the ideal I to abort is the expected behavior as long as a real I would also abort. Here, we make the assumption that any modification to ψ_2 by $\mathcal{A}$ will result in a response that is rejected by a real I. This is true because, by the security of the ring signature scheme, $\mathcal{A}$ cannot produce a valid ring signature as part of the message without access to SK_I^{RS}, SK_R^{RS}, or rsk. Since $\mathcal{A}$ has not corrupted either party in this case, it does not have the capability to generate such a signature except by random guessing (with negligible success). Thus, we can safely assume that I will abort with any modification to ψ_2, and thus the outputs in the simulated case remain indistinguishable from a real interaction.

Note that since we are only considering the actions of $\mathcal{S}$ within the context of a single protocol execution, we do not yet consider the case where $\mathcal{A}$ sends a ψ_2' provided by $\mathcal{Z}$ from another session (i.e., a replay attack). Looking ahead, we will later rule out the possibility of replay attacks by arguing that no information collected from other sessions by $\mathcal{Z}$ can be used to distinguish between challenge protocols.

3.8.4.3 Alteration of ψ_1

In this case, $\mathcal{A}$ alters ψ_1, but does not corrupt either party until the protocol completes. $\mathcal{S}$ begins by generating ψ_1 using NCSim as in the normal case, but then $\mathcal{A}$ alters this flow to be $\psi_1' = \text{“}I\text{”}\|pk'\|npk'\|rpk'\|\sigma_1'\|\sigma_2'$. If ψ_1' has an incorrect format, or any signature is invalid (i.e., $\text{Vrf}(pk', \sigma_1', npk')$ or $\text{Vrf}(pk', \sigma_2', rpk')$ fails to verify), then ψ_1' is ignored by $\mathcal{S}$ and the protocol stalls indefinitely; since $\mathcal{A}$ does not corrupt any parties in this case, the expected behavior of a real session is for R to ignore the invalid prekey. If ψ_1' has the correct format, $\mathcal{S}$ waits until it receives an $(\texttt{establish}, sid, I, R)$ message from $\mathcal{F}_{1psp\text{-}keia}^{\text{IncProc}}$.

This is an example of a case that cannot be simulated without an incriminating abort procedure. $\mathcal{S}$ sends $(\texttt{abort}, sid, I, R)$ to $\mathcal{F}_{1psp\text{-}keia}^{\text{IncProc}}$, but delays delivery of the abort message to I. Next, $\mathcal{S}$ sends $(\texttt{incriminate}, sid, R)$ to $\mathcal{F}_{1psp\text{-}keia}^{\text{IncProc}}$ to gain access to IncProc. $\mathcal{S}$ sends $(\texttt{incriminate}, sid, I, R, pk', npk', rpk')$ to IncProc and receives a response containing $(\texttt{incriminate}, sid, I, R, \psi_2, r^{DRE}, r^{NCE}, r^{RS})$. Immediately, $\mathcal{S}$ sends $(\texttt{set-key}, sid, R, 0)$ to $\mathcal{F}_{1psp\text{-}keia}^{\text{IncProc}}$, causing R to output the same session key k that is encrypted by ψ_2. This is the expected behavior from the real protocol, because a real party R has no way to know if ψ_1 is an authentic message produced by I; it will always output a session key. Note that ideal party I does not produce output because

it has already been marked as "aborted", even though the delayed `abort` message has not yet been delivered by $\mathcal{S}$.

$\mathcal{S}$ then sends the incriminating message ψ_2 through $\mathcal{A}$, which may or may not change it to ψ_2'. If $\mathcal{A}$ allows any response message (whether altered or not) to reach simulated I, then $\mathcal{S}$ delivers the `abort` message to the ideal I. This is the expected result because, in an interactive setting, real party I will always reject any response message that does not contain an encryption of a message including pk. Since $\mathcal{A}$ has altered ψ_1, I will abort on any response from R containing an encryption of $pk' \neq pk$. In the case where $\mathcal{A}$ has chosen $pk' \neq pk$, but then subsequently alters ψ_2' so that it contains an encryption of a message including pk anyway, I will still abort for the same reasons as Case 3.8.4.2 (i.e., $\mathcal{A}$ cannot produce a valid ring signature for ψ_2' with non-negligible probability). Thus, the outputs of the parties are indistinguishable from a real interaction in all cases.

If $\mathcal{A}$ subsequently corrupts R, $\mathcal{S}$ constructs its historical state using information received from IncProc. Concretely, it appears as though R has generated ψ_2 using random coins r^{DRE}, r^{NCE}, and r^{RS}, and the encrypted k matches the output from ideal party R.

If $\mathcal{A}$ corrupts I, $\mathcal{S}$ constructs its historical state by unrigging npk as described in Section 3.8.3, producing key pair (npk, nsk) and random coins r^* such that $(npk, nsk) \leftarrow \text{NCGen}(r^*)$. By the properties of non-committing encryption, these values are indistinguishable from those honestly generated by NCGen during a real protocol session.

3.8.4.4 Initial Corruption of I but not R

When $\mathcal{A}$ chooses to corrupt I before $\mathcal{S}$ has received a `solicit` message from $\mathcal{F}_{1psp\text{-}keia}^{\text{IncProc}}$, the simulator behaves much differently than in the normal cases. Once $\mathcal{S}$ notices that $\mathcal{A}$ causes corrupted party I to broadcast a message ψ_1, $\mathcal{S}$ stops waiting for an ideal party to send a `solicit` message. Since $\mathcal{S}$ always corrupts the corresponding ideal party whenever $\mathcal{A}$ corrupts a simulated party, $\mathcal{S}$ has already corrupted ideal party I when ψ_1 is issued.

Note that if $\mathcal{A}$ instead corrupts an unrelated party P (and $\mathcal{S}$ immediately corrupts ideal party P in response), but then a `solicit` message is sent by I in the ideal setting, then P is not considered to be the session initiator. In other words, the identity of I is always clear from the perspective of $\mathcal{S}$ based on the event that occurs first: either $\mathcal{A}$ generates ψ_1 in the simulated environment, or $\mathcal{S}$ receives a `solicit` message from $\mathcal{F}_{1psp\text{-}keia}^{\text{IncProc}}$.

Once $\mathcal{A}$ has sent ψ_1 from corrupt party I, the ephemeral public keys pk, npk, and rpk are extracted by $\mathcal{S}$ and the signature is verified; if signature verification fails, the message is ignored and the protocol stalls. If ψ_1 is valid, $\mathcal{S}$ causes ideal I to send a `solicit` message to $\mathcal{F}_{1psp\text{-}keia}^{\text{IncProc}}$, marking it as active. Any other messages sent from simulated I by $\mathcal{A}$ are ignored, as expected.

$\mathcal{S}$ waits until the ideal party R sends an `establish` message to $\mathcal{F}_{1psp\text{-}keia}^{\text{IncProc}}$. It then issues a $(\texttt{set-key}, sid, R, 0)$ message to $\mathcal{F}_{1psp\text{-}keia}^{\text{IncProc}}$, causing ideal R to output a session key k. However, since ideal I has been corrupted by $\mathcal{S}$, the `set-key` message normally sent to I is relayed to $\mathcal{S}$ instead, giving the simulator access to k. $\mathcal{S}$ then generates random coins $r^{DRE} \xleftarrow{\$} \{0,1\}^\lambda$, $r^{NCE} \xleftarrow{\$} \{0,1\}^\lambda$, and $r^{RS} \xleftarrow{\$} \{0,1\}^\lambda$. These random coins are used to compute $\gamma \leftarrow \text{NCEnc}(npk, \text{DREnc}(PK_I^{DRE}, PK_R^{DRE}, \text{“}I\text{”}\|pk\|k, r^{DRE}), r^{NCE})$. Since $\mathcal{S}$ has corrupted I, it has access to SK_I^{RS} through $\bar{\mathcal{G}}_{krk}^{\text{Spawn}^*}$. $\mathcal{S}$ makes use of SK_I^{RS} to compute the ring signature $\sigma \leftarrow \text{RSig}(PK_I^{RS}, SK_I^{RS}, \{PK_I^{RS}, PK_R^{RS}, rpk\}, \gamma, r^{RS})$. $\mathcal{S}$ then sends $\psi_2 = \text{“}R\text{”}\|\gamma\|\sigma$ through $\mathcal{A}$.

It does not matter whether $\mathcal{A}$ alters ψ_2 or not, since it has already corrupted I and thus controls its output. In any case, ψ_2 will appear to $\mathcal{A}$ to be a valid message produced by real party R in response to ψ_1. By the security of the ring signature scheme, σ is indistinguishable from a signature produced by an honest party using SK_R^{RS}. Additionally, γ is a valid encryption of k under public key npk, which is consistent with the output of ideal party R. If $\mathcal{A}$ subsequently corrupts simulated R, then $\mathcal{S}$ will reconstruct its historical state using the random coins r^{DRE}, r^{NCE}, and r^{RS}.

3.8.4.5 Early Corruption of R but not I

In this case, $\mathcal{A}$ corrupts R, but not I, before ψ_2 is sent. However, $\mathcal{S}$ does not initially know the identity of R, and thus it generates ψ_1 as if both parties are uncorrupted. As per usual, $\mathcal{S}$ waits until ideal party I sends a `solicit` message to $\mathcal{F}_{1psp\text{-}keia}^{\text{IncProc}}$, then uses NCSim to generate ψ_1 as in Case 3.8.4.1. If $\mathcal{A}$ alters ψ_1, then $\mathcal{S}$ proceeds as in Case 3.8.4.3 (i.e., either $\mathcal{S}$ waits for an `establish` message or the protocol stalls if the format is invalid). In either case, $\mathcal{S}$ is now waiting for an `establish` message.

Before an `establish` message is sent by an ideal party, $\mathcal{A}$ causes the simulated R to output a message ψ_2. $\mathcal{S}$ now knows the identity of R. Note that the behavior of $\mathcal{S}$ is the same if R is corrupted before ψ_1 is sent or after; this case applies to both situations. It also does not matter if $\mathcal{A}$ altered ψ_1 or not; since $\mathcal{A}$ completely controls the behavior of R and it has seen the true ψ_1 value, alteration of ψ_1 has no effect on any party. Now

that R is known to be corrupted, $\mathcal{S}$ unrigs npk as per Section 3.8.3, producing random coins $r^* \xleftarrow{\$} \{0,1\}^\lambda$ such that $(npk, nsk) \leftarrow \text{NCGen}(r^*)$.

As per usual, $\mathcal{A}$ has the ability to alter the message flow containing ψ_2. It does not matter if $\mathcal{A}$ alters this output before it reaches I since $\mathcal{A}$ has complete control of the output when it is first generated. There are two possible cases: either ψ_2 is valid, or it is invalid. We begin by considering the case where ψ_2 is valid.

Although ψ_2 is always sent from R, $\mathcal{A}$ may choose to encode the identity of an arbitrary corrupted party P within ψ_2. It may or may not be the case that $P = R$. Since ψ_2 is valid, it must be of the form $\psi_2 = \text{“}P\text{”}\|\gamma\|\sigma$. $\mathcal{S}$ begins by verifying the ring signature σ. Since $\mathcal{A}$ does not have access to rsk or SK_I^{RS} (as it has not corrupted I), σ is only valid if $\mathcal{A}$ or $\mathcal{Z}$ have access to SK_P^{RS} (due to the security of the ring signature scheme). This can only be the case if P has been corrupted. Since we always assume that parties are corrupted PID-wise (i.e., after corruption, a party remains corrupted during all subsequent sessions), $\mathcal{S}$ can also gain access to SK_P^{RS} by issuing a `retrievesecret` command to $\bar{\mathcal{G}}_{krk}^{\text{Spawn}^*}$.

$\mathcal{S}$ decrypts the non-committing encryption of γ using nsk, and then decrypts the dual-receiver encryption using SK_P^{DRE} (instead of SK_I^{DRE}, as in the honest case). Once $\mathcal{S}$ has decrypted γ, it verifies the correctness of the message (i.e., that it contains the identifier “I” and pk), and extracts k'. Now $\mathcal{S}$ causes ideal party R to send an $(\texttt{establish}, sid, I, R)$ message to $\mathcal{F}_{1psp\text{-}keia}^{\text{IncProc}}$. Next, $\mathcal{S}$ sends $(\texttt{set-key}, sid, I, P, k')$ to $\mathcal{F}_{1psp\text{-}keia}^{\text{IncProc}}$. Since the ideal party R has been corrupted, a `set-key` message is delivered to ideal party I that causes it to output (P, k'). $\mathcal{S}$ does not delay delivery of this message. This is the expected output for a real execution of the protocol, since a corrupted R can correctly identify itself as any party for whom the long-term secret key is known, and R completely controls selection of the shared secret key.

If ψ_2 is not valid (e.g., it is not of the correct form or the ring signature is incorrect), then $\mathcal{S}$ causes ideal party I to abort. It does so by sending an `abort` message to $\mathcal{F}_{1psp\text{-}keia}^{\text{IncProc}}$ and allowing delivery of the `abort` message to ideal party I. This is the expected output in a real execution of the protocol.

If $\mathcal{A}$ subsequently corrupts simulated party I, $\mathcal{S}$ provides r^* as its state history. Due to the security of the non-committing encryption scheme, this state is indistinguishable from a real protocol execution.

3.8.4.6 Post-ψ_1 Corruption of I but not R

In this case, $\mathcal{A}$ does not initially corrupt either party, but it corrupts I after ψ_1 is sent. $\mathcal{A}$ may or may not alter ψ_1. It does not corrupt R until after protocol completion.

Initially, $\mathcal{S}$ proceeds as in the normal case (Case 3.8.4.1), constructing ψ_1 with npk generated by NCSim. Whether $\mathcal{A}$ alters ψ_1 or not, $\mathcal{S}$ normally waits until ideal party R sends an `establish` message to $\mathcal{F}_{1psp\text{-}keia}^{\text{IncProc}}$. In this case, $\mathcal{A}$ corrupts I before such a message is sent. Since $\mathcal{A}$ expects to see an internal state for simulated party I immediately, $\mathcal{S}$ unrigs npk as described in Section 3.8.3, producing r^* such that $(npk, nsk) \leftarrow \text{NCGen}(r^*)$. npk, nsk, and r^* are revealed to $\mathcal{A}$ as the historical state of I.

At this point, $\mathcal{S}$ can proceed as if I was initially corrupted, and npk was chosen by $\mathcal{A}$. In other words, $\mathcal{S}$ can proceed as in Case 3.8.4.4, without making use of the fact that it knows nsk in this case. Specifically, $\mathcal{S}$ will make use of SK_I^{RS}, accessible from $\bar{\mathcal{G}}_{krk}^{\text{Spawn}^*}$ due to corruption of ideal party I, to sign a response ψ_2 from simulated party R. The arguments presented as part of Case 3.8.4.4 apply to this situation, as well.

3.8.4.7 Pre-ψ_2 Corruption of I and R

This situation occurs when $\mathcal{A}$ corrupts both I and R at any point before ψ_2 is sent. If $\mathcal{A}$ corrupts I and causes it to send ψ_1 before $\mathcal{S}$ observes a `solicit` message sent by an ideal party, then $\mathcal{S}$ proceeds as in Case 3.8.4.4. If $\mathcal{A}$ instead waits until after $\mathcal{S}$ has sent a ψ_1 produced with NCSim to corrupt I, then $\mathcal{S}$ unrigs npk as in Case 3.8.4.6. In either case, while $\mathcal{S}$ is waiting for an `establish` message to be sent by an ideal party R, $\mathcal{A}$ causes corrupted simulated party R to send message ψ_2 to I. There is no need for $\mathcal{S}$ to issue any additional messages from the (corrupted) ideal parties I and R since it completely controls both of their outputs. If $\mathcal{A}$ generates output for either corrupted party, $\mathcal{S}$ issues the same output from the corresponding ideal party. Consequently, the output from the ideal protocol is indistinguishable from the output from a real protocol execution.

3.8.4.8 Coda

Note that the cases described in cases 3.8.4.1 through 3.8.4.7 cover all possible situations. Additionally, none of the behaviors described in these cases are contradictory; $\mathcal{S}$ has well-defined behavior under all conditions. For each case, we have shown that

the simulated flows generated by $\mathcal{S}$ are indistinguishable from flows generated in a real execution of Spawn*. These results follow from the security properties of the underlying cryptographic protocols. Additionally, for each case we have shown that the outputs of the ideal parties are equivalent to the outputs of the corresponding parties during a real execution of Spawn*. Therefore, the simulated challenge protocol is indistinguishable from the real challenge protocol within a single session from the joint perspective of $\mathcal{A}$ and $\mathcal{Z}$.

The only remaining condition is that $\mathcal{Z}$ cannot make use of knowledge from other sessions to break the indistinguishability of the current session. The Spawn* protocol has two message flows: ψ_1 and ψ_2. We must show that information from other sessions is not useful for formulating alterations of either message.

ψ_1 is composed entirely of ephemeral public keys (and an unauthenticated assertion of identity), and $\mathcal{Z}$ could instruct $\mathcal{A}$ to alter the message to copy one from another session. However, since we are considering the case where the protocol is being used interactively, I will reject any response from R that makes use of a different ψ_1.[7] Therefore, reuse of a message from another session does not grant $\mathcal{Z}$ any advantage over simply generating its own set of ephemeral keys.

ψ_2 includes pk, the "master" ephemeral public key from ψ_1, as part of its encryption of the session key k. This causes the dual-receiver encryption in ψ_2 to be bound to pk. Additionally, the non-committing encryption of the dual-receiver encryption is created for the key npk. npk is also bound to pk as part of the signature in ψ_1. Likewise, the ring signature includes rpk in the ring, and is also bound to ψ_1. Since all parts of ψ_2 are bound to ψ_1 (except for the identity of the responder, which is public knowledge), information from other sessions are not useful in the construction of this message.

This completes the proof. ■

3.8.5 An Attack on Online Repudiation

When a security model does not capture the complete capabilities of a real-world adversary, it can admit schemes that are not secure in practice. In the proof of Spawn* security, we make an assumption in Case 3.8.4.5 that is not necessarily true in all real-world scenarios. If $\mathcal{Z}$ has previously corrupted some party P and gained access to SK_P^{RS}, it may instruct $\mathcal{A}$ to cause a corrupted party $R \neq P$ to identify itself as P in response to I. In our security model, we assume that since P is corrupted, $\mathcal{S}$ can also access SK_P^{DRE} in order to decrypt

[7] We consider non-interactive use of the protocol in Section 3.8.7.

the response sent by R. When this assumption fails in practice, it yields a potential attack in which $\mathcal{Z}$ can distinguish between simulated and real protocol executions. We will now consider how such an attack might proceed in the real world.

Alice is a whistleblower that has previously provided secret information to Bob, a journalist, using Spawn* over the Internet. Justin is an agent for a group that has some leverage over Bob. Justin wishes to incriminate Alice to prevent further whistleblowing, but he requires evidence that Alice has provided documents to Bob. To accomplish this, Justin demands that Bob asks Alice to send him some new information that will incriminate her. Bob refuses to reveal his long-term secret key SK_B^{RS} to Justin since it would allow Justin to impersonate Bob in all conversations. Justin agrees to this arrangement; Bob will retain control of his secret key, and Justin will instruct Bob to send particular messages to Alice. However, Bob does not actually want to incriminate Alice; instead, he would like to secretly simulate Alice for Justin. If Bob can successfully simulate Alice, then Justin can never incriminate Alice by threatening Bob.

Bob computes a message ψ_1 using NCSim, and reports to Justin that he has received ψ_1 from Alice. However, Justin has covertly stolen the long-term secrets SK_C from Alice's friend Charlie in the past. Justin constructs a message ψ_2 containing session key k that purportedly comes from Charlie, and signs the ring signature using SK_C^{RS}. He then instructs Bob to send ψ_2 to Alice, along with a message, encrypted under k, asking Alice to meet Charlie for coffee (or some other innocuous message). Justin expects Alice to respond to Charlie's message using a protocol that requires her to know k.[8] If Bob has access to SK_C, as we assume in the proof of Spawn* security, then he can forge the expected response from Alice by recovering k from ψ_2. However, if Bob does not have access to SK_C, then he cannot simulate the response; Justin has caught Bob attempting to deceive him.

This attack weakens online repudiation in a nuanced manner. If Justin receives a response purportedly from Alice, then he knows that either Bob was faithfully relaying Justin's messages to the real Alice, or Charlie's secret key was compromised by Bob (and not only by Justin). If Justin does not receive a response, then he knows that either the real Alice did not respond to the (forged) message from Charlie, or Bob has attempted to deceive Justin by simulating Alice. Note that this situation only occurs when Justin "probes" the honesty of Bob by sending a message from Charlie; Justin does not accomplish his primary objective of sending a message to Alice from Bob to provoke incriminating behavior from Alice. If Bob predicts that Justin is going to attempt such a probe, he can easily establish

[8] Although Spawn* is only a key exchange protocol, it is assumed that the shared secret will be used as a key for some overall secure messaging scheme. Any party other than the party that generates k that encrypts a message under k reveals the fact that they were able to decrypt the contents of ψ_2 (and thus that they are in possession of one of the long-term keys required for the dual-receiver encryption).

a true connection to Alice and relay ψ_2 honestly—but if Bob's prediction is incorrect, then he will have no choice but to incriminate Alice. In practice, this means that Justin will always have some uncertainty about the veracity of Alice's responses to Bob. The exact usage of Spawn* within a larger secure messaging solution determines whether or not this attack is considered a problem.

3.8.6 Implications of IncProc

The existence of IncProc is required to handle non-simulatable situations in the proof of security for Spawn*. Specifically, Case 3.8.4.3 requires an incrimination procedure. IncProc perfectly captures the extent to which an adversary can break the deniability of the protocol when all assumptions of the security model hold. For this reason, it is important to consider the implications of IncProc on real-world use of the protocol.

For Spawn*, IncProc causes a valid ψ_2 message to be created and signed using R's secret key SK_R^{RS}. There is a critical difference between Spawn*, Φ_{dre}, and RSDAKE in this respect: when IncProc is invoked in Spawn*, one of the honest parties (namely, R) does not abort. This leads to a potential attack on the deniability of the protocol.

If Mallory, an active network adversary, is attempting to convince Justin that Bob is communicating with Alice using Spawn*, she can do so by exploiting the use of IncProc in the proof of Case 3.8.4.3. Justin begins by generating ψ_1' using $(npk', nsk') \leftarrow \text{NCGen}()$ and sending ψ_1' to Mallory, while ensuring that nsk' is kept private. When Alice sends ψ_1 to Bob, Mallory intercepts this message and replaces it with ψ_1'. Bob responds with ψ_2 containing an encryption γ of session key k and a ring signature σ signed by ring $\{PK_A^{RS}, PK_B^{RS}, rpk\}$. Mallory relays ψ_2 to Justin, but either causes Alice to stall (by never delivering ψ_2 to Alice), or causes Alice to abort (by delivering ψ_2 to Alice). Justin now instructs Mallory to corrupt Bob. When Mallory corrupts Bob (e.g., by confiscating his phone running a secure messaging app using Spawn*), she recovers k'. Mallory sends k' and SK_B to Justin. Justin uses his knowledge of nsk and SK_B^{DRE} to decrypt γ, ensuring that $k = k'$. In this case, Justin will be convinced that Bob attempted to communicate with Alice as long as he believes that Mallory did not corrupt Alice or Bob until after ψ_2 was sent.[9]

[9] If Mallory corrupts either party before ψ_2 is sent, then she has a method for choosing or recovering k, and thus she can forge evidence. Depending on when each party is corrupted, Mallory can use the simulation techniques from Case 3.8.4.4, Case 3.8.4.5, or Case 3.8.4.7 of the security proof. See Table 3.1 for an overview of when the cases apply.

Like the attack described in Section 3.8.5, the setting in which Spawn* is used determines whether or not this attack should be considered a problem.

3.8.7 Non-Interactive Spawn*

The main advantage of Spawn* is that, unlike Φ_{dre} and RSDAKE, it can be used in a non-interactive setting. Before investigating the consequences of using Spawn* non-interactively, we first consider how the two variants might be implemented in practice.

While Spawn* does not depend on any particular networking substrate, it is natural to expect the interactive version to be used in an instant messaging application over the Internet. When Bob wishes to send a message to Alice, he connects to Alice and establishes a TCP/IP connection (perhaps with the assistance of an untrusted relay server for the purpose of NAT traversal). Alice generates and sends ψ_1 to Bob over the connection, to which Bob replies with ψ_2. Alice rejects any response from Bob that makes use of an ephemeral key pk other than the one sent in the context of the TCP connection. Consequently, Alice and Bob enjoy all of the security guarantees of the interactive setting; with the exceptions of the potential attacks in Section 3.8.5 and Section 3.8.6, it is as if Alice and Bob were communicating using $\mathcal{F}_{1psp\text{-}keia}^{\text{IncProc}}$ in the ideal world.[10]

Non-interactive Spawn* is implemented with the assistance of an untrusted central server tasked with the distribution of prekeys. It is natural to consider this version of the protocol in the context of mobile text messaging. Initially, Alice uses her phone's data connection to connect to a central server operated by the developer of a secure communication app using Spawn*. Alice generates a set $\{\psi_{1,1}^{\text{Alice}}, \psi_{1,2}^{\text{Alice}}, ..., \psi_{1,n}^{\text{Alice}}\}$ of n prekeys, where each prekey is a valid message ψ_1 for a Spawn* session. Alice uploads all n prekeys to the central server.[11] Later, Bob wishes to send a message to Alice. Bob uses his phone's data connection to connect to the central server and request a prekey for Alice. The server sends

[10] Given that we list two specific attacks on the security of Spawn*, it is natural to wonder if any other attacks are possible. The attack on deniability described in Section 3.8.6 is specifically admitted by the definition of the ideal functionality and is necessary to GUC-realize an ideal key exchange of this type (i.e., no alternative protocol of the same type could exist without a similar attack). Given the security proof in Section 3.8.4, any other attack must violate an assumption of the security model (as is the case for the attack in Section 3.8.5). Practitioners choosing to use any cryptographic protocol should carefully consider the assumptions of its security model within the context of their specific environment.

[11] In practice, the server might wish to confirm that the prekeys uploaded in Alice's name were actually generated by Alice in order to prevent denial-of-service attacks. A simple way to do this while maintaining all deniability and security properties of the protocol is to use Spawn* or RSDAKE interactively to secure communications between users and the central server.

an available prekey $\psi_{1,i}^{\text{Alice}}$, $1 \leq i \leq n$, to Bob and removes $\psi_{1,i}^{\text{Alice}}$ from its list of available prekeys for Alice.[12] Bob now uses $\psi_{1,i}^{\text{Alice}}$ to compute a response $\psi_{2,i}^{\text{Alice}}$ to complete the key exchange. He sends $\psi_{2,i}^{\text{Alice}}$ along with his message, encrypted with k, to Alice over his text messaging service. Even if Alice is offline at this time, the response will be buffered by store-and-forward servers so that it eventually reaches Alice. Upon receiving the message, Alice locates the corresponding secret keys for the message to recover k.

Unfortunately, this non-interactive capability comes at a cost: the deniability of the protocol is not as strong as the interactive version. Intuitively, the issue is that ψ_1 is no longer part of a single protocol session; it has been moved into a cross-session global infrastructure (i.e., the publicly available prekey distribution server). This breaks a core assumption of the proof provided in Section 3.8.4: the simulator $\mathcal{S}$ can no longer dictate the value of ψ_1, because $\mathcal{Z}$ can reuse ψ_1 values across protocol sessions. If $\mathcal{Z}$ acquires an actual ψ_1' value produced by I as part of an aborted session, then $\mathcal{Z}$ can instruct $\mathcal{A}$ to replace any ψ_1 value generated by $\mathcal{S}$ with ψ_1'. In the interactive setting, a real party I would detect that $\psi_1 \neq \psi_1'$ and abort, providing $\mathcal{S}$ with a simulation strategy. However, I would complete successfully in the non-interactive setting because it has no way to detect that ψ_1' has come from a different session. Consequently, Case 3.8.4.3 no longer holds in the non-interactive setting. In practice, this means that Spawn* does not provide online repudiation when R attempts to simulate I in the non-interactive setting.

As an illustration of the consequences of this weakness, we consider a possible attack against the non-interactive implementation described earlier. After Alice has published $\{\psi_{1,1}^{\text{Alice}}, \psi_{1,2}^{\text{Alice}}, ..., \psi_{1,n}^{\text{Alice}}\}$ to the central server, Justin asks Bob to help incriminate Alice. Bob, not actually willing to incriminate Alice for Justin, attempts to simulate Alice by generating ψ_1 using NCSim and reporting to Justin that he received ψ_1 from the central server as a prekey for Alice. Justin ignores this value and instead requests $\psi_{1,i}^{\text{Alice}}$, an actual prekey for Alice, from the central server. He then generates a session key k and encrypts it according to the Spawn* protocol, producing ciphertext γ encrypted for npk from $\psi_{1,i}^{\text{Alice}}$. Justin then asks Bob to complete $\psi_2 =$ "B"$\|\gamma\|\sigma$ by producing a ring signature σ for γ using his secret key SK_B^{RS}. Bob has no recourse in this situation; he cannot recover k from γ, and thus cannot continue to simulate Alice. Consequently, Justin can be sure that if Bob ever claims to have received a response from Alice encrypted using k, then the response truly did come from Alice. Online repudiation for Bob is lost.

[12] If the server runs out of available prekeys for a user, then that user cannot receive new messages. In practice, the system should ensure that Alice frequently refills her list of available prekeys. This issue is similar to a limitation of Mixminion [DDM03], which makes use of SURBs (single-use reply blocks). If the set of SURBs for a Mixminion user is exhausted, that user cannot anonymously receive packets until the supply is replenished.

Note that the only case of the security proof that is broken in the non-interactive setting is Case 3.8.4.3. Specifically, Alice still maintains online repudiation in this setting—she can reliably simulate a response to her messages from any party, even in the presence of online judges—and all other security properties of the protocol continue to hold. Thus, non-interactive Spawn* still provides stronger deniability guarantees than 3-DH, the current (non-interactive) TextSecure key exchange protocol.

3.8.8 Conjecture: The TextSecure Iron Triangle

Given the incomplete online repudiation of non-interactive Spawn*, an obvious question to ask is whether the protocol can be modified to address these problems. We may also wonder more generally about all key exchanges suitable for use in the TextSecure setting. We define such protocols in the following way:

Definition 3.2 (*TextSecure-like Key Exchanges*)

A *TextSecure-like key exchange* is a one-round key exchange protocol in which the initiator I does not initially know the identity of the responder R.

Our results lead us to an unfortunate suspicion about the nature of such protocols:

Conjecture 3.1 (*TextSecure Iron Triangle*)

Any TextSecure-like key exchange cannot simultaneously provide non-interactivity, forward secrecy[13], and online repudiation with respect to R simulating I.

Intuitively, this conflict arises from the set of secrets required to recover the session key k from the protocol transcript. In general, both I and R may have short-term secrets (sk_I and sk_R, respectively) and long-term secrets (SK_I and SK_R, respectively). In a non-interactive setting, R cannot simulate I's generation of sk_I to an online judge (for the reasons given in Section 3.8.7), and the online judge can insist on generating sk_R itself. Consequently, the only secret information known only by R in this case is SK_R. If R is able to recover k from the transcript, then this implies that the protocol does not have forward secrecy (because only long-term secrets are required to recover k). If R is not able to recover k from the transcript, then this implies that the protocol lacks online repudiation (because R cannot simulate I's subsequent use of k). Additionally, there is also no way to force the judge to reveal any secrets to R since the judge can always insist on the use of a secure multi-party computation protocol to generate any required response.

[13] As we do throughout this section, we refer here to the notion of "weak forward secrecy" defined by Bellare et al. [BPR00].

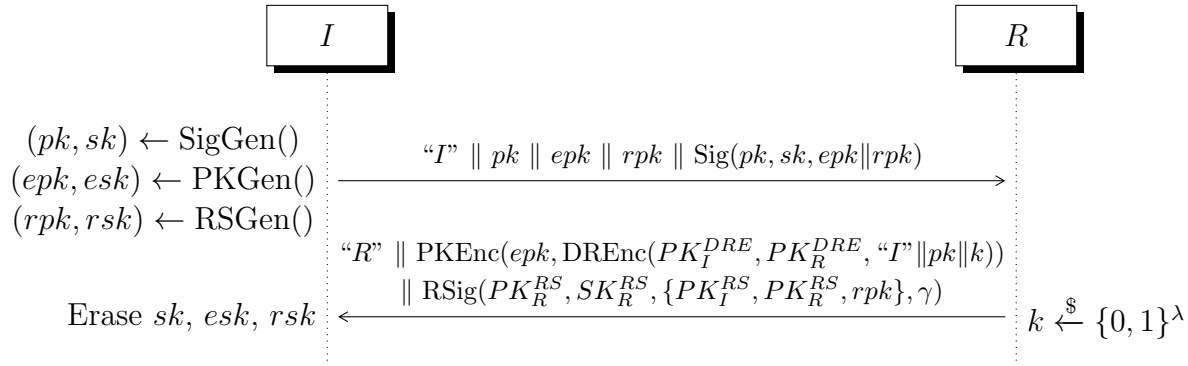


Figure 3.4: Real protocol Spawn. The shared secret is k. γ refers to "R" concatenated with the ciphertext produced by PKEnc.

3.8.9 A Practical Relaxation: Spawn

The proof of Spawn* security in Section 3.8.4 assumes a very strong threat model: the adversary can adaptively corrupt parties, and no information can ever be erased. In practice, these assumptions may not hold. If either assumption is removed, then the security model will admit a modified version of Spawn* with substantially increased performance. Spawn is a protocol that is equivalent to Spawn*, except that it replaces the use of non-committing encryption in ψ_2 with a standard public-key cryptosystem. The motivation for this modification is that non-committing encryption schemes are extremely expensive compared to standard public-key cryptosystems (e.g., the NCE scheme described by Walfish [Wal08] makes $2\lambda n$ calls to the PKEnc function of an underlying 1-bit PKE scheme to encrypt an n-bit message with security parameter λ). Spawn is constructed as shown in Figure 3.4.

By removing the use of non-committing encryption, Spawn is dramatically faster than Spawn* when implemented. The most expensive operation that remains is the use of dual-receiver encryption. We now extend the proof in Section 3.8.4 to show that Spawn maintains the same security properties as Spawn* when adaptive corruptions are disallowed, or when memory can be securely erased and only semi-adaptive corruptions are allowed.

Theorem 3.4 (*Security of Spawn with Static Corruption*)

Assuming the existence of a signature scheme (SigGen, Sig, Vrf), ring signature scheme (RSGen, RSig, RVrf) secure under full-key exposure, a dual-receiver encryption scheme (DRGen, DREnc, DRDec), and a public-key cryptosystem (PKGen, PKEnc, PKDec), Spawn GUC-realizes $\mathcal{F}_{1psp\text{-}keia}^{\text{IncProc}}$ within the $\bar{\mathcal{G}}_{krk}^{\text{Spawn}^*}$-hybrid model with no erasures and static security when IncProc proceeds as in Algorithm 5.

Proof: The proof given in Section 3.8.4 also applies here if all uses of non-committing encryption are replaced with public-key encryption. All uses of NCGen, NCEnc, and NCDec are replaced by PKGen, PKEnc, and PKDec, respectively. All that remains is to explain how $\mathcal{S}$ behaves when it would normally make use of NCSim and NCEqv.

In all cases of the proof (except Case 3.8.4.4), $\mathcal{S}$ uses NCSim to produce npk, for use in ψ_1, and γ, for later use as part of ψ_2. In our case, $\mathcal{S}$ instead uses PKGen to generate (epk, esk). ψ_1 is constructed by including epk and signing it with pk. Where $\mathcal{S}$ normally uses γ generated by NCSim, it instead generates random coins $r \leftarrow \{0,1\}^\lambda$ and computes $\gamma \leftarrow \text{PKEnc}(epk, \text{DREnc}(PK_I^{DRE}, PK_R^{DRE}, \text{“}I\text{”}\|pk\|r))$. By the security of the public-key cryptosystem, a γ generated in this way is indistinguishable from an honestly generated encryption $\text{PKEnc}(epk, \text{DREnc}(PK_I^{DRE}, PK_R^{DRE}, \text{“}I\text{”}\|pk\|k))$ when esk is unknown.

Of course, $\mathcal{S}$ can no longer make use of NCEqv. There are two cases in the proof where NCEqv is used: Case 3.8.4.1 and Case 3.8.4.2. In both cases, $\mathcal{A}$ corrupts either I or R after ψ_2 has been sent. Since we do not allow for adaptive corruptions, these situations do not apply. ■

Theorem 3.5 (*Security of Spawn in the Semi-Adaptive Erasure Model*)

Assuming the existence of a signature scheme (SigGen, Sig, Vrf), ring signature scheme (RSGen, RSig, RVrf) secure under full-key exposure, a dual-receiver encryption scheme (DRGen, DREnc, DRDec), and a public-key cryptosystem (PKGen, PKEnc, PKDec), Spawn GUC-realizes $\mathcal{F}_{1psp\text{-}keia}^{\text{IncProc}}$ within the erasure $\bar{\mathcal{G}}_{krk}^{\text{Spawn}^*}$-hybrid model with semi-adaptive security when IncProc proceeds as in Algorithm 5.

Proof: The proof is identical to the proof of Theorem 3.4, with the exception of the rationale for the replacement of NCEqv. There are two cases in the proof where NCEqv is used: Case 3.8.4.1 and Case 3.8.4.2. In both cases, $\mathcal{A}$ corrupts either I or R after ψ_2 has been sent. In the erasure model, it is assumed that I securely erases esk after it has received ψ_2. Since we only allow semi-adaptive corruptions, $\mathcal{A}$ is not permitted to corrupt I until it has either been aborted or has output a key. In either case, esk has already been securely deleted, and thus $\mathcal{Z}$ cannot decrypt the previously observed ψ_1 value; it remains indistinguishable from an honestly generated message. ■

In practice, it is reasonable to accept this weaker threat model in many common environments. Security and privacy tools such as hard drive encryption utilities and secure messaging tools commonly assume that cryptographic keys can be limited to RAM storage,

and RAM can be securely erased while a machine is uncorrupted. It is also reasonable to assume that corruptions are not fully adaptive. When Spawn is used interactively, I can easily erase esk if a timely response is not received from R (e.g., if an adversary prevents delivery of ψ_2 in an attempt to cause I to retain esk for later corruption). In the non-interactive setting, online repudiation of Spawn* is already weakened (see Section 3.8.5). Since an online judge can always mandate the use of a valid prekey from I by retrieving it from the central server, there are no practical situations in non-interactive Spawn* that actually require the use of NCEqv for simulation. For this reason, use of Spawn instead of Spawn* causes no loss of deniability beyond that already incurred due to use in a non-interactive environment.

3.8.10 Spawn as an Axolotl Bootstrap

Since the ultimate goal of Spawn is to improve the deniability properties of TextSecure, it must be able to replace the 3-DH key exchange used to initiate Axolotl. Spawn cannot be used as a drop-in replacement because it is a non-contributory key exchange, while 3-DH is a contributory protocol. It is also important to avoid attempting to modify Spawn to make it contributory, because the existence of ephemeral secrets for I will destroy the deniability properties of the scheme. Instead, we will demonstrate how Spawn (or Spawn*) can be used to bootstrap Axolotl, providing all deniability benefits of the key exchange while also achieving per-message forward and backward secrecy.

The Axolotl specifications [Per13] require I and R to exchange long-term identity keys and ephemeral ratchet keys. I sends identity key A, and R sends identity key B. I sends ephemeral keys $(A0, A1)$, and R sends ephemeral keys $(B0, B1)$. The two parties then derive a shared secret `master_key` from $(A, A0)$ and $(B, B0)$ using 3-DH. From `master_key`, a key derivation function is used to compute a variety of keys used for message transmission. $A1$ and $B1$ are used as initial contributions within the DH ratchet.

In the general Axolotl specification, the roles of initiator and responder are not initially known. However, we assume that these roles are initially known by the parties (as is the case for TextSecure). This allows us to remove the requirement for sending $A1$, which is only used when the roles are determined after the initial key exchange. Additionally, we can remove the need for $A0$ and $B0$ by changing the way that the master key is computed. This is trivially accomplished by using the Spawn protocol in the initial exchange, and setting `master_key` $\leftarrow k$. We can also remove the explicit requirement for $B1$ to be included in the initial key exchange by setting `ratchet_flag` $\leftarrow$ True for R (i.e., requiring R to generate a fresh ephemeral ratchet key as part of the first message). Now Axolotl can continue as normal.

Intuitively, R will send initial messages to I using a key ratchet based solely on the secret key k exchanged by Spawn. R will also include a DH ephemeral key as part of its messages. When I responds, it will include a DH ephemeral key that completes the initial key exchange. At this point, the scheme has been ratcheted forward to a new set of chain keys that does not use k for message encryption in any way, but message authentication is still traceable to the initial key exchange (through the use of root keys for the key derivation function).

3.9 Selecting a Protocol

The best choice of key exchange protocol for a secure messaging scheme is highly dependent on the environment in which the scheme will be used. Φ_{dre}, RSDAKE, and Spawn have different security guarantees and usability properties, and thus are best suited for different environments.

Spawn is the only protocol that supports non-interactive environments. It also requires the fewest number of flows—only two messages are needed to complete the key exchange. Although non-interactive Spawn does not provide online repudiation with respect to R simulating I (see Section 3.8.7), it still provides improved deniability properties compared to 3-DH. Despite the existence of the security weaknesses described in Section 3.8.5 and Section 3.8.6, Spawn is also useful in interactive environments where these weaknesses are not a concern.

In interactive settings, Φ_{dre} and RSDAKE can also be used. Φ_{dre} and RSDAKE offer the same security properties, but RSDAKE offers some additional features; RSDAKE is a contributory key exchange that has been proven to be secure in the post-specified peer setting.

To select an appropriate protocol for use in a given environment, it is also important to understand how the schemes perform in practice under various network conditions. In Chapter 4, we perform a comparative performance evaluation of the protocols in order to provide concrete advice for practitioners seeking to make use of the schemes.

Chapter 4

Implementation

In Chapter 2, we described a disconnect between secure messaging system developers and the academic community. Specifically, there is an abundance of solutions described in the literature that are never implemented. It has become clear that describing a new system and writing security proofs, while necessary, are insufficient for making cryptography usable; we need to do more if we want actual users to benefit from our schemes. One way to bridge this gap is to provide open implementations of our designs to encourage use by developers of consumer security products. As part of this work, we developed open-source implementations of every scheme presented in Chapter 3.[1] This chapter provides an overview of the implementations, as well as a comparative evaluation of the schemes.

4.1 Overview

When implementing cryptographic libraries, choosing an appropriate programming language is extremely important. Ideally, the programs produced using the language should be highly efficient, since the speed of cryptographic operations is typically bound by processor speed. Moreover, it is important that the language does not encourage the introduction of security vulnerabilities; cryptographic libraries are often used in environments where they are placed under heightened scrutiny by attackers and where security is a paramount concern. For these reasons, our implementations were produced using the Go programming language [Go 09]. Go is a compiled and strongly typed language with a variety of integrated cryptographic libraries.

[1] The resulting libraries, as well as other software projects maintained by the CrySP research group, can be found at https://crysp.uwaterloo.ca/software/.

Our primary development objective was to implement the four key exchange schemes described in Chapter 3: Φ_{dre} with non-interactive DRE, Φ_{dre} with interactive DRE, RSDAKE, and Spawn. Unfortunately, the specialized cryptosystems used by these protocols lack widely available implementations. Consequently, we also developed implementations of these underlying cryptosystems as part of our overall development effort. In summary, we produced the following libraries as part of this work:

- Section 4.2.1: Pairing-Based Cryptography Library [Lyn06] wrapper for Go;
- Section 4.2.2: Shacham-Waters [SW07] ring signatures;
- Section 4.2.3: HORS [RR02] one-time signatures with HORS+ [ZMM10] improvement;
- Section 4.2.4: Cramer-Shoup cryptosystem [CS98], both in prime order finite fields and elliptic curve groups;
- Section 4.2.5: Chow-Franklin-Zhang [CFZ14] BDDH-based dual-receiver encryption;
- Section 4.2.6: Interactive dual-receiver encryption (defined in Section 3.6.3);
- Section 4.2.7: Φ_{dre} [Wal08];
- Section 4.2.8: RSDAKE (defined in Section 3.7.2);
- Section 4.2.9: Spawn (defined in Section 3.8.9).

All of the implemented schemes are provably secure in the standard model (i.e., they do not require random oracles). For each scheme, we make note of the security assumptions made by the associated proofs. We evaluate the performance of the schemes in Section 4.3.

4.2 Libraries

4.2.1 PBC Go Wrapper

Several of the schemes required to construct the higher-level protocols make use of pairing-based cryptography. Cryptographic pairings are defined over three mathematical groups: G_1, G_2, and G_T, where each group is of the same order r. Additionally, a bilinear map e maps a pair of elements—one from G_1 and another from G_2—to an element in G_T.

Given two generators $g \in G_1$ and $h \in G_2$, the map e has the property that $e(g^x, h^y) = e(g, h)^{xy}$ for any $x, y \in \mathbb{Z}_r$. This property can be exploited to produce a variety of efficient cryptosystems, such as those used to construct our DAKEs.

Despite the power of pairings, there are very few active implementations of pairing-based cryptography. The two most prominent projects are the Pairing-Based Cryptography Library (PBC), originally authored by Ben Lynn [Lyn06], and the RELIC toolkit [AG09]. We elected to construct our schemes using PBC since it provides a well-documented implementation of pairings over elliptic curve groups of composite order (i.e., where the order of the group is the product of two large primes), and this setting is required for our chosen ring signature scheme.

To make use of PBC in our projects, the library was first ported to the Microsoft Visual Studio environment for Windows (previously, PBC required MinGW for Windows compilations). We then developed a wrapper for the library to expose all of its functionality to Go. The wrapper adds type checking to operations, as well as automatic garbage collection and integration with the standard Go libraries.

4.2.2 Ring Signatures

Ring signatures, described in Section 3.4.6, are required to implement both RSDAKE and Spawn. For our implementation, we chose to implement the ring signature scheme proposed by Shacham and Waters [SW07]. This ring signature scheme provides anonymity even in the event of full-key disclosure, and signatures are unforgeable even in the presence of malicious insiders. These security properties hold in the standard model with three complexity assumptions: integer factorization, computational Diffie-Hellman in prime-order cyclic subgroups of elliptic curves, and subgroup decision in composite-order elliptic curves. The subgroup decision problem, with respect to a multiplicative cyclic group of order $n = pq$ (where p and q are prime) having subgroup G_q of order q, is defined by Shacham and Waters in the following way [SW07]:

Definition 4.1 (*The Subgroup Decision Problem*)

> Given w selected at random either from G (with probability 1/2) or from G_q (with probability 1/2), decide whether w is in G_q. For this problem one is given the description of G, but *not* the factorization of n.

Our implementation of the Shacham-Waters scheme is based on the PBC Go wrapper. The order n of the pairing groups is generated using the standard Go RSA key generator. Several predefined configurations are provided for users, offering security levels between 80

and 256 bits. Depending on the selected security level, the scheme internally makes use of the SHA-256 or SHA-512 hash functions on messages before signing them.

4.2.3 One-Time Signatures

One-time signatures are a type of digital signature that can only be used to sign a single message. In exchange for this concession, the resulting schemes are extremely fast and often exhibit stronger security properties than traditional digital signature schemes. For example, one-time signatures can be strongly unforgeable using only standard model assumptions.[2] One-time signature schemes can be used to improve the performance of our higher-level protocols. They are also required in order to maintain the security of our chosen DRE scheme.

We chose to implement the HORS one-time signature scheme proposed by Reyzin and Reyzin [RR02]. HORS is extremely efficient and produces strongly unforgeable signatures in the presence of one-time adaptive chosen message attacks. HORS is similar to a family of schemes derived from the original Lamport signature scheme [Lam79]. These approaches are based on releasing a large number of cryptographic hashes as a public key, and then subsequently releasing selected preimages based on the message to be signed. HORS decreases the size of signatures by taking a cryptographic hash of the message and then using substrings of the hash to determine which preimages to release. The security of HORS is based on the one-wayness and "subset-resilience" of the hash function.[3]

Since "subset-resilience" is not a well-studied property of hash functions, an improvement to HORS was proposed by Zhang, Ma, and Moon [ZMM10]. The resulting scheme, HORS+, alters the manner in which hashes are used to select preimages to release. HORS+ provides the same security properties as HORS, but relies only on the one-wayness and collision resistance of the underlying hash function.

Our implementation of HORS+ provides a variety of predefined security settings offering between 80 and 256 bits of security. We selected parameters that provide a balanced trade-off between signature size and computational demands. Depending on the selected

[2] A signature scheme is strongly unforgeable if it is existentially unforgeable and, given signatures on some message m, the adversary cannot produce a new signature on m [BSW06].

[3] A hash H function is "subset-resilient" if, given a message m_1, it is infeasible to find another message m_2 such that that $H(m_2)$ produces a subet of $H(m_1)$ when the hashes are partitioned into binary representations of set indices.

security level, the internal hash is either SHA-256, SHA-512, or SHAKE256.[4] Messages signed with the library are first randomized using the NIST SP 800-106 procedure [Dan09]. This procedure generates a nonce that is mixed with the message before signing—it is intended to protect against attacks that occur when the entity selecting the message is not the same as the entity signing the message.

4.2.4 Cramer-Shoup

The Cramer-Shoup public-key cryptosystem [CS98] is an efficient scheme that is IND-CCA2 secure using only standard model assumptions.[5] The proof of security for the scheme assumes only the hardness of the decisional Diffie-Hellman problem in the underlying group, and the collision resistance of the underlying hash function. These reasons led us to suggest Cramer-Shoup as the basis for the interactive dual-receiver encryption scheme described in Section 3.6.3. We implemented the Cramer-Shoup scheme in two settings: prime order finite fields, and elliptic curves.

Our prime order finite field implementation makes use of SHA-256 or SHA-512 as the underlying hash, based on the selected security level. The parameters for the field are generated using the NIST FIPS 186-4 standard [Com13]. Allowable key sizes are extended beyond the range advised by FIPS 186-4 to include those in the NIST SP 800-57 recommendation [Com12]. Messages are uniquely mapped to group elements for encryption by squaring them to obtain a quadratic residue; they are subsequently mapped back to the message space by finding the square root. We specifically select groups with order 3 (mod 4) so that square roots can be efficiently computed using Lagrange's method.

Our elliptic curve implementation uses SHA-256 or SHA-512 as the underlying hash, and the elliptic curves over prime fields defined in the NIST FIPS 186-4 standard [Com13]. Encryption in this implementation makes use of a hybrid scheme. Cramer-Shoup is used to transmit a secret random curve point. The X coordinate of this point is hashed to key an AES-256 cipher in GCM mode; this symmetric cipher is then used to transmit the actual message.

We compared the two approaches to select an implementation for use in constructing the higher-level schemes. On a single 3.6 GHz core, the prime order finite field implementation requires approximately 100 milliseconds to generate a key, encrypt a message, and decrypt

[4] SHAKE256 is a hash in the SHA-3 family that provides a configurable security level. We use SHAKE256 in the case where we require at least 256 bits of security, but we need more than 256 bits of hash output in order to satisfy HORS' restrictions on parameters.

[5] Refer to Section 3.4.3 for a detailed definition of the IND-CCA2 security game.

the resulting ciphertext at the 128-bit security level. The elliptic curve implementation requires approximately 25 milliseconds to perform the same operations. Consequently, we made use of the elliptic curve implementation in subsequent constructions.

4.2.5 Non-Interactive DRE

Both Φ_{dre} and Spawn make use of dual-receiver encryption, described in Section 3.4.4. We chose to implement the non-interactive DRE scheme proposed by Chow, Franklin, and Zhang [CFZ14]. While these authors construct several schemes in their publication, we do not require the complete non-malleability properties of their more expensive schemes; instead, we implemented the approach based on assumed hardness of the bilinear decisional Diffie-Hellman (BDDH) problem. This scheme provides completeness, soundness, symmetry, public verifiability, and dual-receiver IND-CCA1 security without the use of random oracles.[6]

Our implementation makes use of the HORS+ one-time signature scheme described in Section 4.2.3, as well as the PBC Go wrapper. The dual-receiver IND-CCA1 security of the Chow-Franklin-Zhang scheme relies on the fact that all ciphertexts make use of different verification keys for the underlying digital signatures. For this to be true in the IND-CCA1 setting, it is insufficient for the signature scheme to be existentially unforgeable; the signature scheme must be strongly unforgeable. Consequently, only one-time signature schemes can be used to realize the DRE cryptosystem.

In total, the implementation relies on five hardness assumptions: discrete logarithms in prime-order elliptic curve groups, discrete logarithms in finite fields, BDDH, and the one-wayness and collision resistance of the underlying hash used by HORS+.

Similarly to the elliptic curve Cramer-Shoup implementation described in Section 4.2.4, our dual-receiver encryption implementation makes use of a hybrid encryption scheme. The Chow-Franklin-Zhang protocol is used to encrypt a random element in the pairing's target group G_T, which is a point in $\mathbb{F}_q^2$. The coordinates of this point, which are independently selected uniformly at random, are used to directly construct a symmetric key. This symmetric key is used to encrypt the actual message using AES-256 in GCM mode.

[6] Refer to Section 3.4.4 for definitions of dual-receiver encryption security properties.

4.2.6 Interactive DRE (IDRE)

Our interactive dual-receiver encryption (IDRE) implementation was constructed exactly as described in Section 3.6.3. Concretely, we made use of the elliptic curve Cramer-Shoup implementation described in Section 4.2.4 to produce two encryptions of the message. An interactive zero-knowledge proof of knowledge is then generated to show that both ciphertexts are valid and that they encrypt the same plaintext. The resulting scheme relies only on the hardness assumptions made by the underlying Cramer-Shoup scheme.

The implementation also includes the capability to perform a non-interactive encryption using the Fiat-Shamir heuristic [FS87]; this facility is not used in our constructions because it requires the random oracle model to prove its security.

4.2.7 Φ_{dre}

Our implementation of Φ_{dre}, originally proposed by Walfish [Wal08] and described in Section 3.6, makes use of the relaxed version that replaces non-committing encryption with standard public-key encryption. It is acceptable to make use of the relaxed version if we assume that memory can be erased; our implementation does not store ephemeral keys beyond the key exchange session. The implementation uses the elliptic curve Cramer-Shoup implementation described in Section 4.2.4 to encrypt the session key. Either the non-interactive DRE scheme described in Section 4.2.5 or the interactive DRE scheme described in Section 4.2.6 can be used to encrypt the messages under the long-term keys (but the choice of scheme must be fixed before the long-term keys are generated). We refer to the former scheme as Φ_{dre} and to the latter scheme as Φ_{idre}. The security of the implementation relies on the assumptions of the chosen DRE scheme and the elliptic curve Cramer-Shoup scheme. To our knowledge, ours is the first implementation of Φ_{dre}; neither the original thesis written by Walfish [Wal08] nor the subsequent publication by Dodis et al. [DKSW09] mention an implementation or performance measurements.

4.2.8 RSDAKE

We implemented RSDAKE, described in Section 3.7.2, using the Shacham-Waters ring signature scheme (see Section 4.2.2). The ephemeral signing key pair for each party is generated using the Elliptic Curve Digital Signature Algorithm (ECDSA) described in the NIST FIPS 186-4 standard [Com13]. The key exchange procedure is completed using Elliptic Curve Diffie-Hellman. To minimize the necessary security assumptions, the ring

signature scheme, signature scheme, and Diffie-Hellman scheme all operate over the same curves defined in the ECDSA standard. The identifiers of the parties are encoded as 8-byte sequences; it is presumed that these identifiers would be assigned as part of a higher-level protocol. The security of the implementation relies only on the security assumptions for the underlying schemes.

4.2.9 Spawn

We implemented Spawn, described in Section 3.8.9, in an interactive setting for the purpose of comparative evaluation. Like our implementation of RSDAKE, our implementation of Spawn uses ECDSA to generate ephemeral signing key pairs. Ring signatures are produced using the Shacham-Waters scheme. The shared secret key is encrypted using the elliptic curve Cramer-Shoup scheme. The security of the implementation relies only on the security assumptions for the underlying schemes.

4.3 Evaluation

To compare the performance of the key exchange implementations, we instantiated a simulation of an interactive session between two parties over the Internet. This simulation modeled a duplex connection with configurable transmission latency and bandwidth. We evaluated the performance of four protocols: Φ_{dre} using the non-interactive Chow-Franklin-Zhang DRE, Φ_{idre}, RSDAKE, and Spawn. We tested each protocol in a variety of simulated network conditions at 112-, 128-, and 192-bit approximate security levels. We simulated message latencies at 0, 50, 100, 300, 1000, 2000, 5000, and 10000 milliseconds. We simulated communication channel bandwidth at 10 Gib/s, 100 Mib/s, 20 Mib/s, 5 Mib/s, 500 Kib/s, and 50 Kib/s. We performed each test 200 times on 3.6 GHz processor cores with access to RAM providing 15 GiB/s read and write speeds with 63 ns latency.

All graphs in this section make use of logarithmic vertical axes and error bars. The error bars, which denote the standard error of the mean, are typically too small to see.

4.3.1 Space Complexity

All four schemes transmit different amounts of data during the protocol session. The amount of data transmitted depends only on the choice of protocol and the security level;

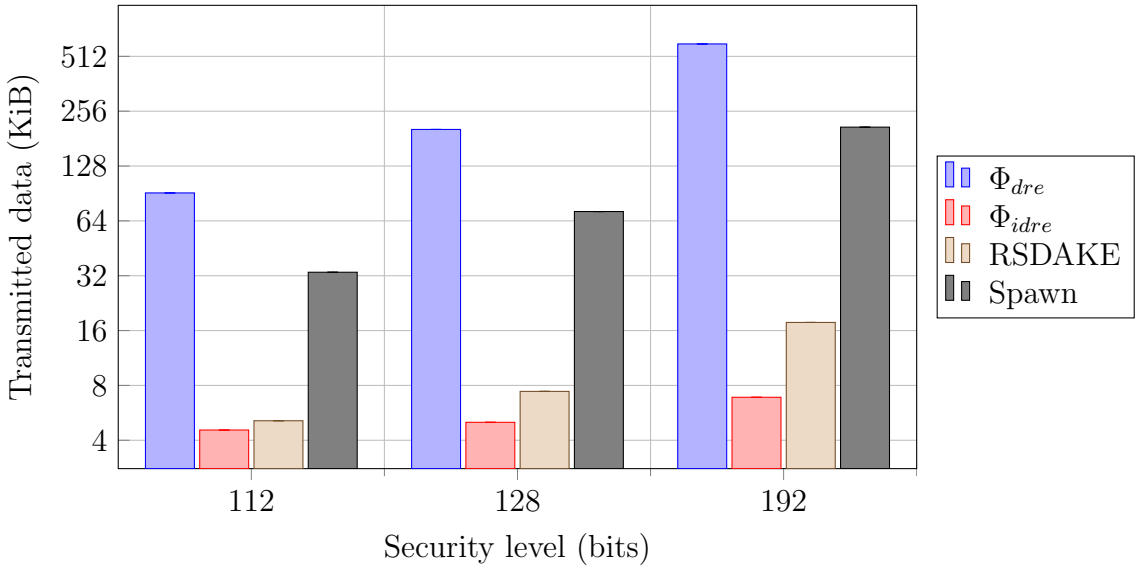


Figure 4.1: The amount of data transmitted increases significantly with higher security levels. Φ_{dre} and Spawn require significantly more transmissions than Φ_{idre} or RSDAKE.

it does not depend on the speed of the network connection. Figure 4.1 shows the total amount of data transmitted by each protocol during a session; this total represents the sum of the number of bytes written at the application layer by each party.

All schemes are relatively expensive compared to a simple SIGMA protocol of the form described in Section 3.7; all four schemes require at least 4 KiB to complete a session with at least 112 bits of security. However, both Φ_{dre} and Spawn require significantly more data transmission than Φ_{idre} or RSDAKE; additionally, Φ_{dre} transmits approximately three times more data than Spawn. The reason for this disparity is the use of the HORS+ one-time signature scheme by the non-interactive DRE implementation. The Chow-Franklin-Zhang DRE scheme requires that an ephemeral public key and signature for a one-time signature scheme are transmitted as part of every encryption. These keys and signatures are extremely large because they contain large sets of hashes and preimages. Since Φ_{dre} makes use of three DRE encryptions and Spawn makes use of only one, Φ_{dre} requires nearly three times more data than Spawn to complete the exchange. The use of interactive DRE in Φ_{idre} dramatically reduces the data costs of the protocol; Φ_{idre} consistently uses the least data of all four protocols.

While RSDAKE uses nearly as little data as Φ_{idre} for the 112-bit security level, its costs increase much faster as the security level is increased. Since the Shacham-Waters ring

signature scheme used by RSDAKE requires composite-order bilinear groups, the relative ease of the integer factorization problem requires that the size of group elements increases with approximately the cube of the security level. As a result, the two ring signatures exchanged within RSDAKE rapidly grow in size with heightened security. Nonetheless, RSDAKE never approaches the transmission costs of Φ_{dre} or Spawn, even at the 192-bit security level.

4.3.2 Time Complexity vs. Security Level

As the desired security level increases, all four schemes require increasingly expensive cryptographic operations. To understand the impact of security levels on the time complexity of the algorithms, we focus on the simulation with minimal impact from network conditions. Figure 4.2 shows the amount of time required to complete a session of each protocol when the parties are connected through a 10 Gib/s channel with no latency; the resulting delays are directly indicative of the cryptographic overhead associated with each scheme.

Several interesting observations can be drawn from Figure 4.2. RSDAKE and Spawn are roughly an order of magnitude more expensive than Φ_{dre} and Φ_{idre}. The ring signature scheme used by both RSDAKE and Spawn is to blame for this disparity. As we mentioned in Section 4.3.1, the Shacham-Waters scheme used in our implementation makes use of composite-order bilinear groups. Operations in this group setting are considerably more expensive than operations in the prime-order elliptic curve groups used by Φ_{dre} and Φ_{idre}. RSDAKE suffers more from this expense than Spawn does; RSDAKE makes use of two ring signatures while Spawn only uses one.

Both Φ_{dre} and Φ_{idre} are extremely computationally efficient, requiring less than one second to complete at the 112- and 128-bit security levels. However, in this fast network environment, the interactive DRE scheme used by Φ_{idre} scales better than the Chow-Franklin-Zhang scheme used by Φ_{dre}. At the 128- and 192-bit security levels, Φ_{idre} requires the least amount of time to complete. This performance improvement can be attributed to the direct use of elliptic curve groups by the Cramer-Shoup scheme in Φ_{idre}, rather than the use of pairing-based cryptography in Φ_{dre}.

While Figure 4.2 compares the performance of the schemes under ideal network conditions, it is also useful to understand how the schemes react to poor network conditions. Figure 4.3 shows the total time required to complete a session of each protocol when the parties are communicating across a 50 Kib/s connection with 2 seconds of latency. This simulation models an extremely poor network environment that is effectively a worst-case

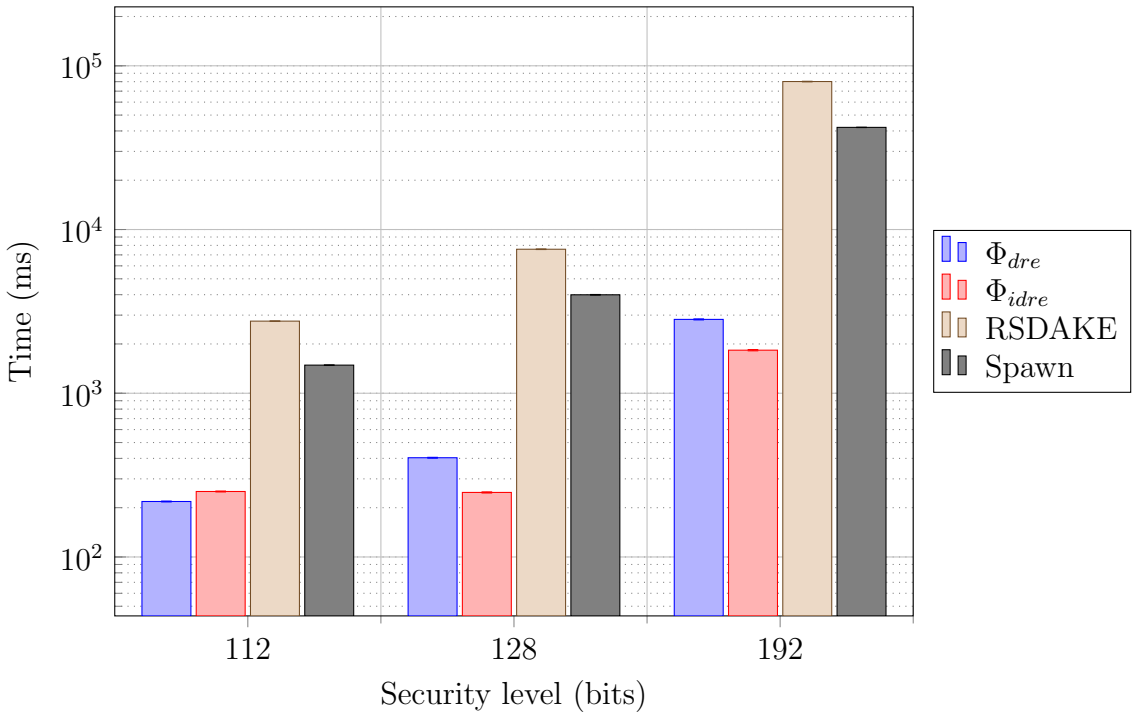


Figure 4.2: Over a high-bandwidth connection with no latency, the cryptographic overhead of each protocol is clear. The use of ring signatures negatively affects the performance of RSDAKE and Spawn.

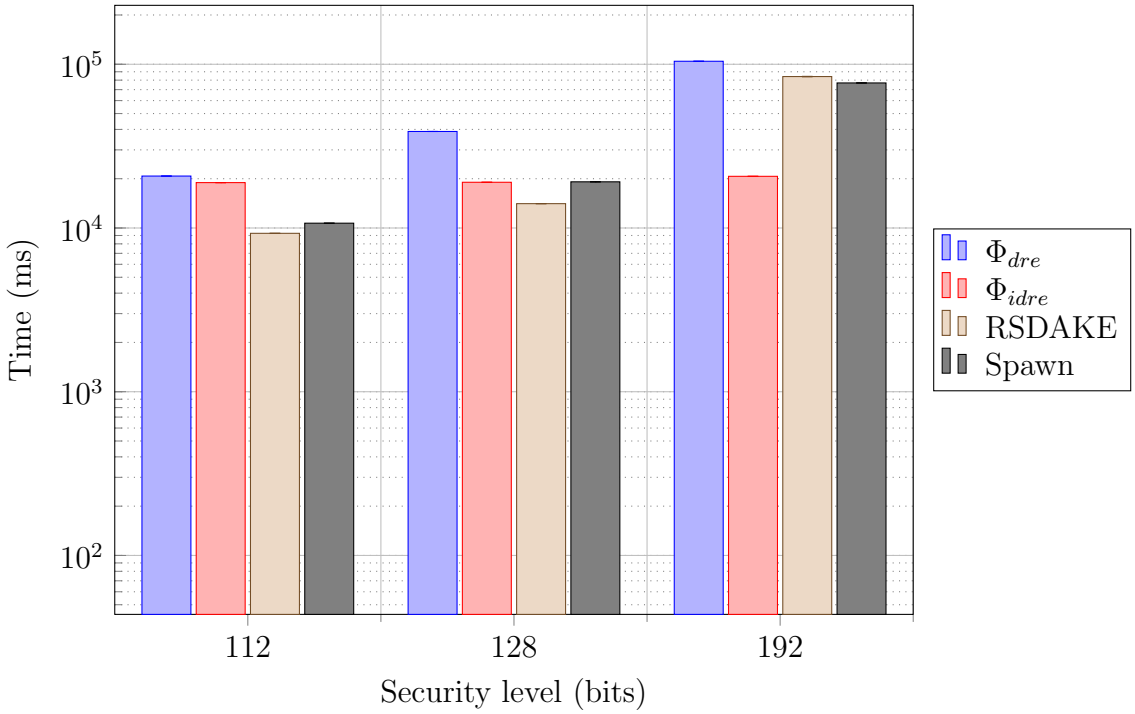


Figure 4.3: When communicating over a high-latency and low-bandwidth channel, the limitations of the network begins to affect protocol performance. RSDAKE and Spawn perform the best at 112- and 128-bit security levels.

scenario for the protocols; the primary use of this model is to provide insight into how the protocols behave under difficult network conditions.

The most immediate observation that can be made from Figure 4.3 is that the poor performance of the network connection dominates the cost of all four protocols; all protocols take approximately 10 seconds to complete, even at the 112-bit security level. Despite requiring nine message flows to complete, Φ_{idre} performs comparatively well in this high-latency environment. Φ_{idre} is the most scalable protocol; since it only requires operations in small elliptic curve groups, the cryptographic overhead is relatively constant. Φ_{dre} performs the worst at all security levels since it makes use of the Chow-Franklin-Zhang DRE scheme three times, which imposes high bandwidth costs. Spawn generally performs well since it only requires two message flows to complete, but it is still slower than RSDAKE at 128- and 192-bit security levels due to its use of the Chow-Franklin-Zhang DRE scheme. RSDAKE is the most efficient protocol at the 112- and 128-bit security levels. At the 192-bit security level, the performance of RSDAKE and Spawn is impacted by the computational costs

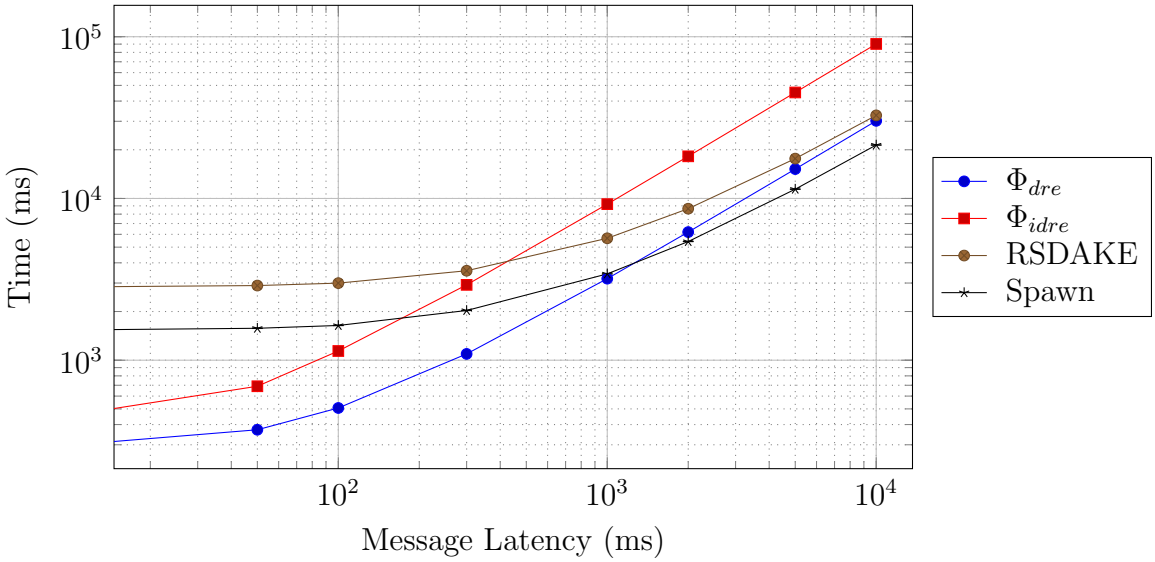


Figure 4.4: As message latency increases, protocols at the 112-bit security level are grouped by the number of messages they send.

of the Shacham-Waters ring signature scheme. Consequently, Φ_{idre} performs significantly better than all other protocols at the 192-bit security level.

4.3.3 Time Complexity vs. Latency

To understand the effect of network latency on the performance of the protocols, we examined the total time required to complete protocol sessions over a high-bandwidth channel with varying latency. Figure 4.4 plots the performance of the four protocols at the 112-bit security level as the latency is increased. When the latency is low, the performance of the algorithms approaches the condition depicted in Figure 4.2. As the latency increases, the time required to complete the session becomes dominated by the performance of the communication channel. At the highest level of latency, the protocols are effectively grouped by the number of flows that they require: Φ_{idre}, which requires nine message flows, performs the worst; Φ_{dre} and RSDAKE, which both require three message flows, perform similarly; and Spawn, which only requires two message flows, performs the best. Consequently, as the latency is increased, the protocols scale at different rates.

The effect of message latency on the performance of the protocols changes as the security level increases. Figure 4.5 and Figure 4.6 plot the performance of the protocols at the

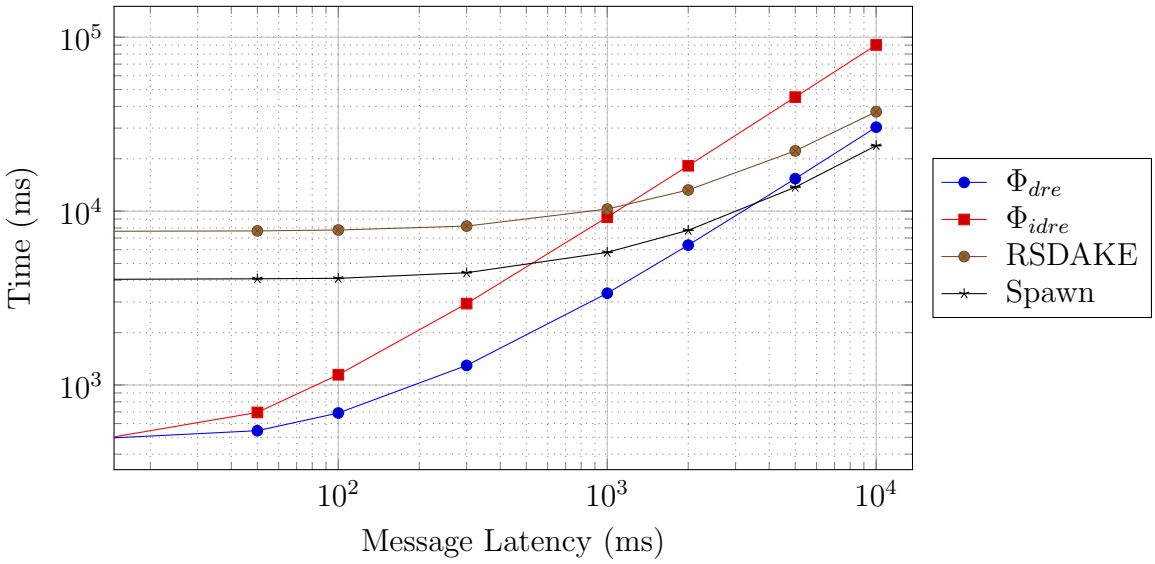


Figure 4.5: At the 128-bit security level, the higher computational costs of RSDAKE and Spawn are more significant than message latency.

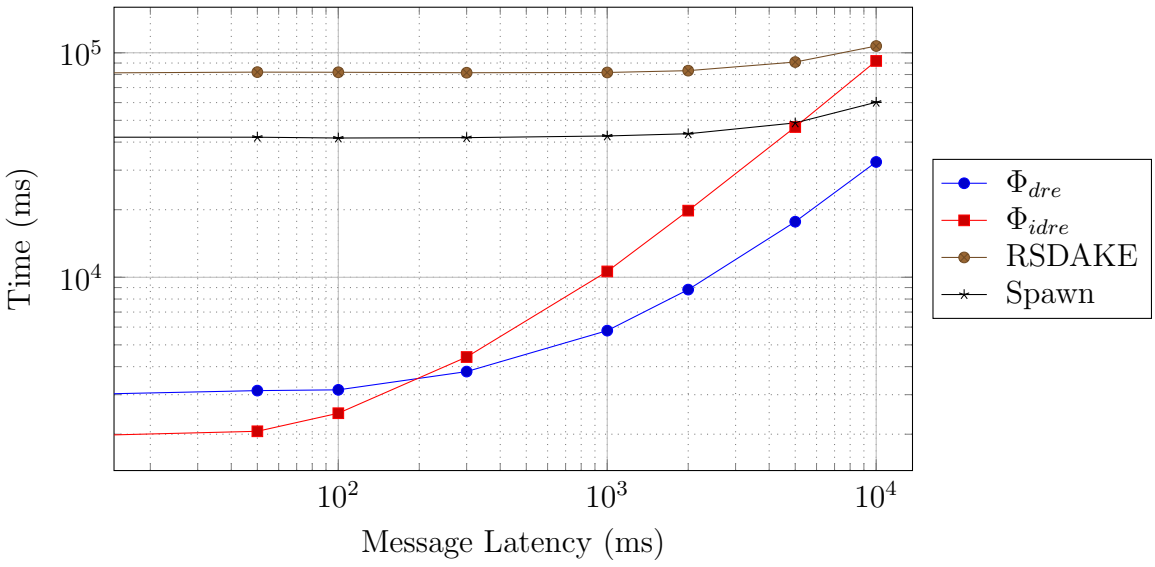


Figure 4.6: At the 192-bit security level, the computational costs of RSDAKE and Spawn effectively eliminate their performance advantages with respect to message latency.

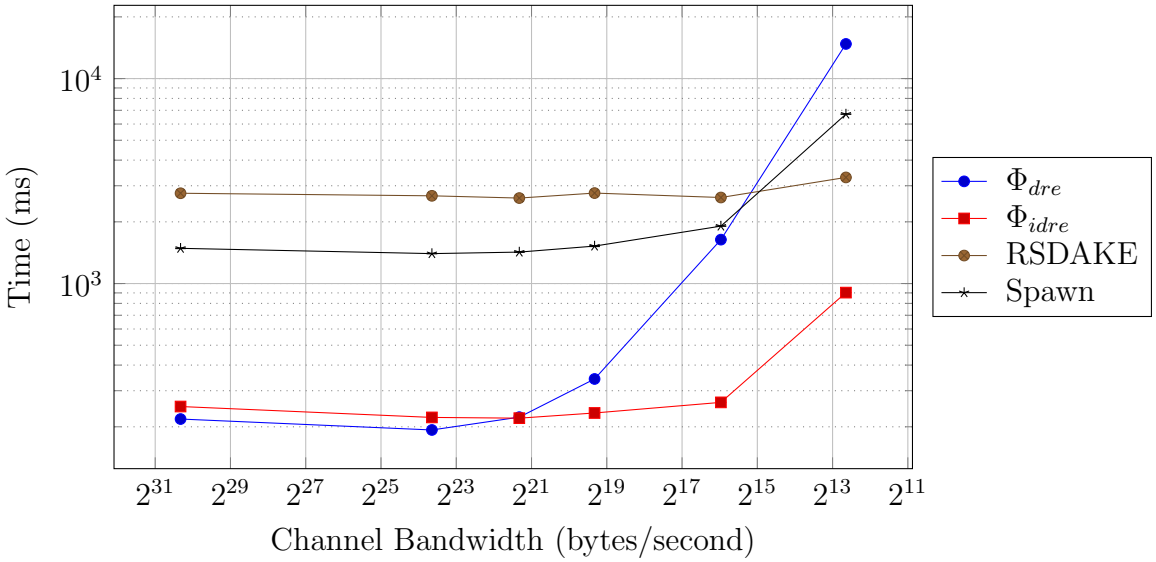


Figure 4.7: At the 112-bit security level, Φ_{idre} performs the best in low-bandwidth environments. Φ_{dre} scales particularly poorly due to its high data transmission requirements.

128-bit and 192-bit security levels, respectively. As the security level increases, the high computational costs of the Shacham-Waters ring signature scheme begins to dominate the performance costs of RSDAKE and Spawn. While Spawn still achieves the best performance at the 128-bit security level in environments with extremely high latency, these advantages are eliminated at the 192-bit security level; the performance of RSDAKE and Spawn at this security level is almost completely independent of the message latency. In contrast, Φ_{dre} and Φ_{idre} continue to scale in a similar manner. Notably, the performance of Φ_{idre} is still significantly harmed by increasing network latency since it requires the most message flows.

4.3.4 Time Complexity vs. Bandwidth

When considering the impact of network conditions on key exchange protocols, it is also important to consider the effect of bandwidth constraints. To measure this effect, we evaluated the total time required to complete each protocol session in a network environment with no latency under varying bandwidth constraints.

Figure 4.7 plots the performance of the four protocols at the 112-bit security level as the available bandwidth decreases. Note that the horizontal axis in this plot is reversed:

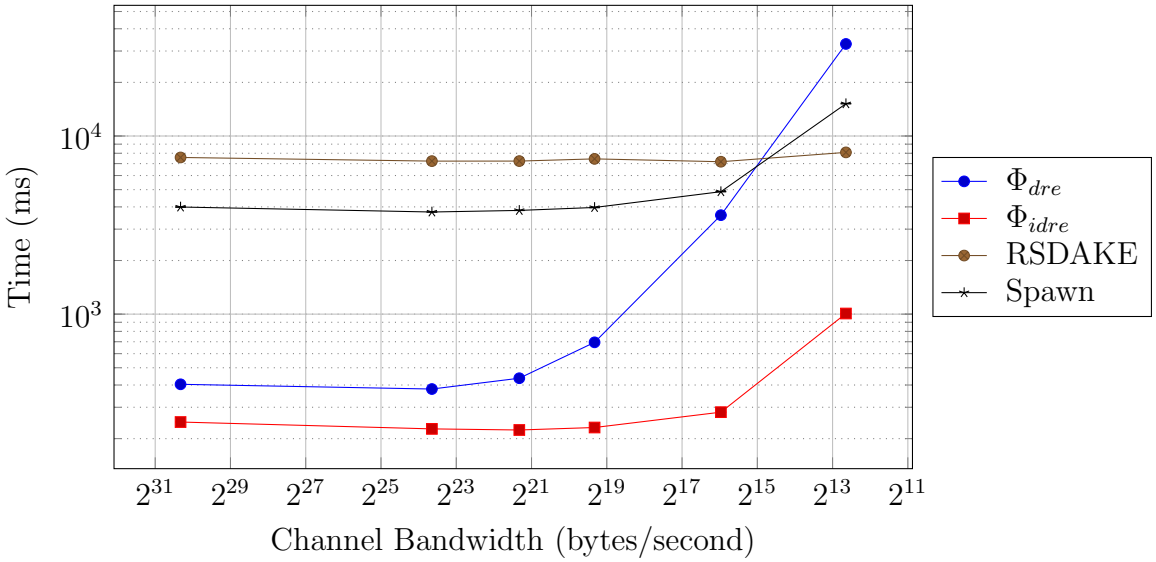


Figure 4.8: At the 128-bit security level, the dominance of Φ_{idre} becomes more pronounced. The performance of Spawn begins to become dominated by the cost of the ring signature scheme.

the bandwidth of the channel decreases toward the right side of the figure. When the communication channel supports 20 Mib/s (2.5 MiB/s) or more, the performance of the protocols is effectively constant; this represents the situation depicted in Figure 4.2. However, as the available bandwidth is decreased, the communication costs quickly dominate the performance of each scheme. The rate at which the performance of each protocol decreases is based on the amount of data that is transmitted; consequently, the performance of the schemes in the lowest-bandwidth environment is effectively predicted by the data transmission totals in Figure 4.1. When there is very little available bandwidth, Φ_{idre} becomes the most efficient scheme. RSDAKE also performs very well in this low-bandwidth environment since it requires a relatively small amount of data to be transmitted. The performance of RSDAKE in Figure 4.7 is relatively constant with respect to varying bandwidth constraints because its performance is dominated by the computational costs of the ring signature scheme.

When configuring the protocols for higher security levels, they still scale in a similar manner with respect to bandwidth constraints. Figure 4.8 and Figure 4.9 plot total session time against channel bandwidth for protocols offering 128 and 192 bits of security, respectively. Similarly to Section 4.3.3, we note that the performance of RSDAKE and Spawn becomes dominated by the costs of the Shacham-Waters ring signature scheme at

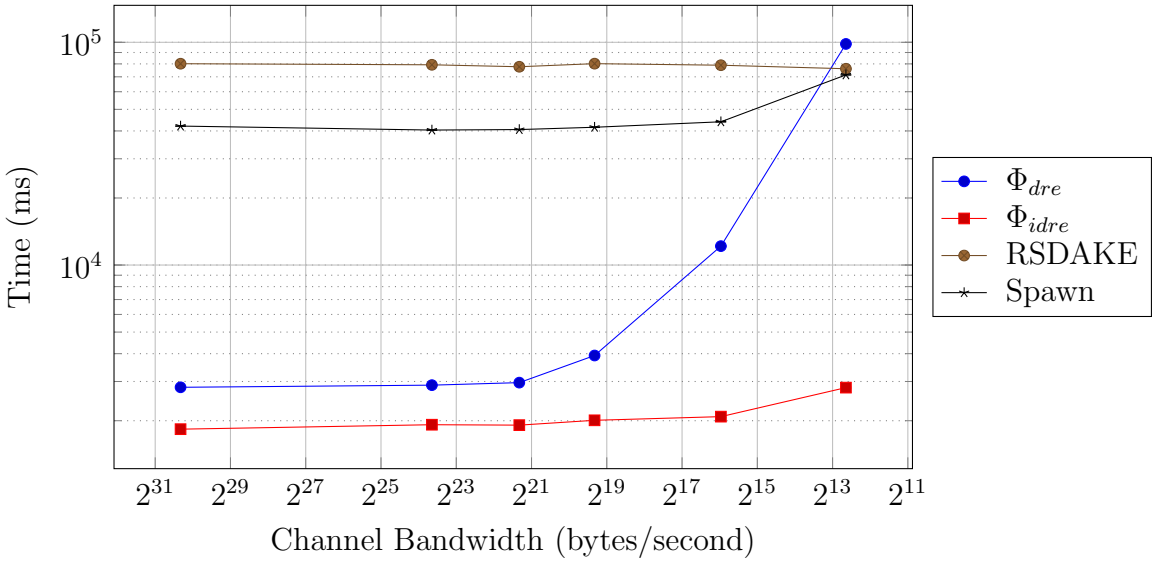


Figure 4.9: At the 192-bit security level, Φ_{idre} solidifies its performance advantage. RSDAKE and Spawn are relatively unaffected by changing bandwidth conditions.

these higher security levels. In particular, the performance of Spawn is only affected by bandwidth constraints at the 192-bit security level in the most extreme network environments. Φ_{dre} performs consistently poorly in low-bandwidth environments under all security levels, since it is the scheme that requires the largest data transmissions. Φ_{idre} performs the fastest under all network conditions at the 128- and 192-bit security levels due to its computationally efficient construction and minimal data transmission requirements.

4.4 Discussion

When choosing a protocol to use in a real-world application, developers should consider both their security and performance needs. We previously discussed the security characteristics of Φ_{dre}, Φ_{idre}, RSDAKE, and Spawn in Section 3.9. Our evaluation presented in Section 4.3 is meant to serve as a guideline for real-world performance expectations; the final performance of a scheme depends on the implementation and the underlying cryptosystems used to construct it.

In environments where the amount of transmitted data is the primary concern, such as mobile data connections that charge based on monthly upload and download totals,

Φ_{idre} and RSDAKE are the best choices. If the speed of the key exchange is the primary concern, then the best selection depends on the expected network conditions and desired security level. When parties communicate over connections with low latency and high bandwidth, Φ_{dre} and Φ_{idre} are the best choices. In contrast, Φ_{dre} should be avoided in environments with low bandwidth, and Φ_{idre} should be avoided in environments with high latency. RSDAKE and Spawn are superior in environments with both high latency and low bandwidth. When computational demands are the primary concern, RSDAKE and Spawn should be avoided due to their use of relatively expensive ring signature schemes, especially at higher security levels. If a developer would like to build a system that performs well on different kinds of devices across multiple interactive environments, they could dynamically select a DAKE using an initial client handshake protocol; TLS uses a similar negotiation scheme for establishing secure connections [DR08]. However, if the system will primarily be used in non-interactive environments, then only Spawn should be used.

If ring signature schemes become more efficient than dual-receiver cryptosystems in the future, then RSDAKE and Spawn have the potential to outperform the Φ_{dre} and Φ_{idre} schemes. Additionally, all of the implementations described in this chapter make use of only standard-model assumptions. If a practitioner is willing to make use of schemes that depend on random oracles for security, then the performance of all four protocols can be greatly improved.

Chapter 5

Concluding Remarks

In this work, we sought to provide assistance for practitioners seeking to implement deniable secure messaging protocols. We systematized knowledge of existing approaches, providing much-needed context amidst the current development fervor. We decomposed secure messaging protocols into three aspects (trust establishment, conversation security, and transport privacy), easing the task of analyzing and constructing these complex systems. We then focused specifically on the deniability properties offered by the conversation security layer. We examined existing deniable authenticated key exchange (DAKE) protocols in the context of multiple definitions of deniability. We introduced several new DAKEs (RSDAKE and Spawn*) and variants (Φ_{idre} and Spawn) that are well-suited to different secure messaging environments. Most notably, we have introduced Spawn, the first protocol with strong deniability properties and forward secrecy that can be used in non-interactive environments. We quantified our claims of practicality by comparing instantiations of the protocols in simulated network environments, and released the resulting code to the public.

Of course, much work remains to be done. The world still lacks a small set of usable secure messaging tools with strong and well-defined security properties. Reaching this goal will require close collaboration between theorists and practitioners, as well as widespread agreement on desired protocol properties. Additionally, there remain open research problems, such as protection of metadata, that still lack good solutions. It is our hope that the systematization presented in Chapter 2 will help to inform these developments.

Deniability of secure messaging schemes also remains a research area with many unsolved problems. The most appropriate definition of deniability to use when constructing protocols is not yet agreed upon; specifically, very few publications consider online repudiation during analysis of their designs. While we suspect that weak forward secrecy and

online repudiation are mutually exclusive properties in the non-interactive setting, Conjecture 3.1 remains unproven. Finally, although we provide proof-of-concept implementations of our new DAKE protocols, adoption by end-user tools may be encouraged by integrating these implementations with a higher-level popular cryptographic library.

Since it is easy to become ensnared by the details of security proofs or the intricacies of an implementation, it is important that we do not lose sight of the bigger picture. The recent surveillance revelations have reminded us all of the lack of security and privacy in our digital communications. Currently, end users are confused by the wide selection of tools promising secure messaging, none of which is perfect, and some of which are broken or malicious. While the general public debates the politics of government surveillance, these debates are only meaningful if technology empowers the people with a choice. It is our responsibility as security researchers to provide this choice through the design and construction of usable and effective secure messaging tools.

References

[ACMP10] Michel Abdalla, Céline Chevalier, Mark Manulis, and David Pointcheval. "Flexible Group Key Exchange with On-Demand Computation of Subgroup Keys". In: *Progress in Cryptology–AFRICACRYPT*. Springer, 2010, pp. 351–368.

[AG07] Chris Alexander and Ian Goldberg. "Improved User Authentication in Off-The-Record Messaging". In: *Workshop on Privacy in the Electronic Society*. ACM. 2007, pp. 41–47.

[AG09] Diego de Freitas Aranha and Conrado Porto Lopes Gouvêa. *RELIC is an Efficient LIbrary for Cryptography*. 2009. URL: https://github.com/relic-toolkit/relic (visited on 2015-04-13).

[AJM95] Richard Ankney, Don Johnson, and Stephen Mike Matyas. *The Unified Model*. Contribution to X9F1. 1995.

[AKJ+15] Bonnie Brinton Anderson, C Brock Kirwan, Jeffrey L Jenkins, David Eargle, Seth Howard, and Anthony Vance. "How Polymorphic Warnings Reduce Habituation in the Brain—Insights from an fMRI Study". In: *CHI*. ACM. 2015.

[And97] Ross Anderson. "Two Remarks on Public Key Cryptology". Available from https://www.cl.cam.ac.uk/users/rja14. 1997.

[BBL02] Ian Brown, Adam Back, and Ben Laurie. *Forward Secrecy Extensions for OpenPGP*. Draft. Internet Engineering Task Force, 2002. URL: https://tools.ietf.org/id/draft-brown-pgp-pfs-03.txt.

[BBS04] Dan Boneh, Xavier Boyen, and Hovav Shacham. "Short Group Signatures". In: *Advances in Cryptology–CRYPTO*. Springer. 2004, pp. 41–55.

[BDG14] Nikita Borisov, George Danezis, and Ian Goldberg. *DP5: A Private Presence Service*. Tech. rep. 2014-10. CACR, 2014.

[BDS+03] Dirk Balfanz, Glenn Durfee, Narendar Shankar, Diana Smetters, Jessica Staddon, and Hao-Chi Wong. "Secret Handshakes from Pairing-Based Key Agreements". In: *Symposium on Security and Privacy*. IEEE. 2003, pp. 180–196.

[BGB04] Nikita Borisov, Ian Goldberg, and Eric Brewer. “Off-the-Record Communication, or, Why Not To Use PGP”. In: *Workshop on Privacy in the Electronic Society*. ACM. 2004, pp. 77–84.

[BGK08] Alexandra Boldyreva, Vipul Goyal, and Virendra Kumar. “Identity-based Encryption with Efficient Revocation”. In: *Conference on Computer and Communications Security*. ACM. 2008, pp. 417–426.

[BGKT12] Michael Backes, Ian Goldberg, Aniket Kate, and Tomas Toft. “Adding Query Privacy to Robust DHTs”. In: *Symposium on Information, Computer and Communications Security*. ACM. 2012, pp. 30–31.

[BHvS12] Joseph Bonneau, Cormac Herley, Paul C. van Oorschot, and Frank Stajano. “The Quest to Replace Passwords: A Framework for Comparative Evaluation of Web Authentication Schemes”. In: *Symposium on Security and Privacy*. IEEE. San Francisco, CA, USA, 2012. URL: http://www.jbonneau.com/doc/BHOS12-IEEESP-quest_to_replace_passwords.pdf.

[Bit12] Bitmessage Project. *Bitmessage*. 2012. URL: https://bitmessage.org/ (visited on 2014-11-02).

[BKM06] Adam Bender, Jonathan Katz, and Ruggero Morselli. “Ring Signatures: Stronger Definitions, and Constructions without Random Oracles”. In: *Theory of Cryptography*. Springer, 2006, pp. 60–79.

[Blo99] Eric A Blossom. *The VP1 Protocol for Voice Privacy Devices Version 1.2*. Communication Security Corporation, 1999.

[BMG+07] Kevin Bauer, Damon McCoy, Dirk Grunwald, Tadayoshi Kohno, and Douglas Sicker. “Low-Resource Routing Attacks Against Tor”. In: *Workshop on Privacy in the Electronic Society*. ACM. 2007, pp. 11–20.

[BMHD08] Zhang Bin, Feng Meng, Xiong Hou-ren, and Hu Dian-you. “Design and Implementation of Secure Instant Messaging System Based on MSN”. In: *International Symposium on Computer Science and Computational Technology*. Vol. 1. IEEE. 2008, pp. 38–41.

[BMP04] Colin Boyd, Wenbo Mao, and Kenneth G Paterson. “Key Agreement using Statically Keyed Authenticators”. In: *Applied Cryptography and Network Security*. Springer, 2004, pp. 248–262.

[BPR00] Mihir Bellare, David Pointcheval, and Phillip Rogaway. “Authenticated Key Exchange Secure Against Dictionary Attacks”. In: *Advances in Cryptology–EUROCRYPT*. Springer. 2000, pp. 139–155.

[BR93] Mihir Bellare and Phillip Rogaway. “Entity Authentication and Key Distribution”. In: *Advances in Cryptology–CRYPTO’93*. Springer, 1993, pp. 232–249.

[BST01] Fabrice Boudot, Berry Schoenmakers, and Jacques Traoré. "A fair and efficient solution to the socialist millionaires' problem". In: *Discrete Applied Mathematics* 111.1 (2001), pp. 23–36.

[BST07] Jiang Bian, Remzi Seker, and Umit Topaloglu. "Off-the-Record Instant Messaging for Group Conversation". In: *International Conference on Information Reuse and Integration*. IEEE. 2007, pp. 79–84.

[BSW06] Dan Boneh, Emily Shen, and Brent Waters. "Strongly Unforgeable Signatures Based on Computational Diffie-Hellman". In: *Public Key Cryptography*. Springer, 2006, pp. 229–240.

[Can01] Ran Canetti. "Universally Composable Security: A New Paradigm for Cryptographic Protocols". In: *Foundations of Computer Science*. IEEE. 2001, pp. 136–145.

[Can04] Ran Canetti. "Universally Composable Signature, Certification, and Authentication". In: *Computer Security Foundations Workshop*. IEEE. 2004, pp. 219–233.

[CDF+99] John Callas, Lutz Donnerhacke, Hal Finney, David Shaw, and Rodney Thayer. *OpenPGP Message Format*. RFC 4880 (Proposed Standard). Updated by RFC 5581. Internet Engineering Task Force, 1999. URL: http://tools.ietf.org/rfc/rfc4880.txt.

[CDMW09] Seung Geol Choi, Dana Dachman-Soled, Tal Malkin, and Hoeteck Wee. "Improved Non-Committing Encryption with Applications to Adaptively Secure Protocols". In: *Advances in Cryptology–ASIACRYPT 2009*. Springer, 2009, pp. 287–302.

[CDNO97] Ran Canetti, Cynthia Dwork, Moni Naor, and Rafi Ostrovsky. "Deniable Encryption". In: *Advances in Cryptology–CRYPTO*. Springer, 1997, pp. 90–104.

[CF10] Henry Corrigan-Gibbs and Bryan Ford. "Dissent: Accountable Anonymous Group Messaging". In: *Conference on Computer and Communications Security*. ACM. 2010, pp. 340–350.

[CF11] Cas Cremers and Michéle Feltz. *One-round Strongly Secure Key Exchange with Perfect Forward Secrecy and Deniability*. Tech. rep. 2011/300. Cryptology ePrint Archive, 2011. URL: https://eprint.iacr.org/2011/300.

[CFGN96] Ran Canetti, Uri Feige, Oded Goldreich, and Moni Naor. *Adaptively Secure Multiparty Computation*. Tech. rep. Massachusetts Institute of Technology, 1996. URL: http://theory.csail.mit.edu/ftp-data/pub/people/oded/dynamic.ps.

[CFZ14] Sherman SM Chow, Matthew Franklin, and Haibin Zhang. "Practical Dual-Receiver Encryption. Soundness, Complete Non-Malleability, and Applications". In: *Topics in Cryptology–CT-RSA 2014*. Springer, 2014, pp. 85–105.

[CGM+11] Sandy Clark, Travis Goodspeed, Perry Metzger, Zachary Wasserman, Kevin Xu, and Matt Blaze. "Why (Special Agent) Johnny (Still) Can't Encrypt: A Security Analysis of the APCO Project 25 Two-way Radio System". In: *Security Symposium*. USENIX, 2011.

[Cha88] David Chaum. "The Dining Cryptographers Problem: Unconditional Sender and Recipient Untraceability". In: *Journal of Cryptology* 1.1 (1988), pp. 65–75.

[CHK03] Ran Canetti, Shai Halevi, and Jonathan Katz. "A Forward-Secure Public-Key Encryption Scheme". In: *Advances in Cryptology–EUROCRYPT*. Springer, 2003, pp. 255–271.

[CK02] Ran Canetti and Hugo Krawczyk. "Security Analysis of IKE's Signature-Based Key-Exchange Protocol". In: *Advances in Cryptology–CRYPTO 2002*. Springer, 2002, pp. 143–161.

[CKFP10] Joseph A Cooley, Roger I Khazan, Benjamin W Fuller, and Galen E Pickard. "GROK: A Practical System for Securing Group Communications". In: *International Symposium on Network Computing and Applications*. IEEE. 2010, pp. 100–107.

[Com12] U.S. Department of Commerce / National Institute of Standards & Technology. *Recommendation for Key Management–Part 1: General*. Version 3. 2012.

[Com13] U.S. Department of Commerce / National Institute of Standards & Technology. *Digital Signature Standard (DSS)*. 2013.

[Con14] Confide. *Confide - Your Off-the-Record Messenger*. 2014. URL: https://getconfide.com/ (visited on 2014-11-02).

[CR03] Ran Canetti and Tal Rabin. "Universal Composition with Joint State". In: *Advances in Cryptology–CRYPTO 2003*. Springer, 2003, pp. 265–281.

[CS98] Ronald Cramer and Victor Shoup. "A Practical Public Key Cryptosystem Provably Secure against Adaptive Chosen Ciphertext Attack". In: *Advances in Cryptology–CRYPTO'98*. Springer. 1998, pp. 13–25.

[Cv13] Jeremy Clark and Paul C van Oorschot. "SoK: SSL and HTTPS: Revisiting past challenges and evaluating certificate trust model enhancements". In: *Symposium on Security and Privacy*. IEEE. 2013, pp. 511–525.

[CWF12] Henry Corrigan-Gibbs, David Isaac Wolinsky, and Bryan Ford. *Proactively Accountable Anonymous Messaging in Verdict*. Tech. rep. arXiv:1209.4819. Version Extended Version. arXiv e-prints, 2012.

[Dan09] Quynh H Dang. *Randomized Hashing for Digital Signatures*. U.S. Department of Commerce / National Institute of Standards & Technology, 2009.

[Dav01] Don Davis. “Defective Sign & Encrypt in S/MIME, PKCS#7, MOSS, PEM, PGP, and XML”. In: *Annual Technical Conference, General Track*. USENIX, 2001, pp. 65–78.

[DDM03] George Danezis, Roger Dingledine, and Nick Mathewson. “Mixminion: Design of a Type III Anonymous Remailer Protocol”. In: *Symposium on Security and Privacy*. IEEE. 2003, pp. 2–15.

[DDN98] Danny Dolev, Cynthia Dwork, and Moni Naor. “Non-Malleable Cryptography”. In: *SIAM Journal on Computing*. 1998, pp. 542–552.

[DGK06] Mario Di Raimondo, Rosario Gennaro, and Hugo Krawczyk. “Deniable Authentication and Key Exchange”. In: *Conference on Computer and Communications Security*. ACM. 2006, pp. 400–409.

[DH76] Whitfield Diffie and Martin E Hellman. “New Directions in Cryptography”. In: *Transactions on Information Theory* 22.6 (1976), pp. 644–654.

[DKSW09] Yevgeniy Dodis, Jonathan Katz, Adam Smith, and Shabsi Walfish. “Composability and On-Line Deniability of Authentication”. In: *Theory of Cryptography*. Springer, 2009, pp. 146–162.

[DMS04] Roger Dingledine, Nick Mathewson, and Paul Syverson. *Tor: The Second-Generation Onion Router*. Tech. rep. DTIC, 2004.

[DN00a] Ivan Damgård and Jesper Buus Nielsen. “Improved Non-Committing Encryption Schemes Based on a General Complexity Assumption”. In: *Advances in Cryptology–CRYPTO 2000*. Springer. 2000, pp. 432–450.

[DN00b] Cynthia Dwork and Moni Naor. “Zaps and Their Applications”. In: *Foundations of Computer Science*. IEEE. 2000, pp. 283–293.

[DNS98] Cynthia Dwork, Moni Naor, and Amit Sahai. “Concurrent Zero-Knowledge”. In: *Symposium on Theory of Computing*. ACM. 1998, pp. 409–418.

[DR08] Tim Dierks and Eric Rescorla. *The Transport Layer Security (TLS) Protocol Version 1.2*. RFC 5246 (Proposed Standard). Internet Engineering Task Force, 2008. URL: http://tools.ietf.org/rfc/rfc5246.txt.

[DvW92] Whitfield Diffie, Paul C van Oorschot, and Michael J Wiener. “Authentication and Authenticated Key Exchanges”. In: *Designs, Codes and Cryptography* 2.2 (1992), pp. 107–125.

[EDG09] Nathan S Evans, Roger Dingledine, and Christian Grothoff. “A Practical Congestion Attack on Tor Using Long Paths”. In: *Security Symposium*. USENIX, 2009, pp. 33–50.

[EFL+99] Carl Ellison, Bill Frantz, Butler Lampson, Ron Rivest, Brian Thomas, and Tatu Ylonen. *SPKI Certificate Theory*. RFC 2693 (Experimental). Internet Engineering Task Force, 1999. URL: http://tools.ietf.org/rfc/rfc2693.txt.

[Ele14] Electronic Frontier Foundation. *Secure Messaging Scorecard*. 2014. URL: https://www.eff.org/secure-messaging-scorecard (visited on 2014-11-11).

[Ell96] Carl M Ellison. "Establishing Identity Without Certification Authorities". In: *Security Symposium*. USENIX, 1996, pp. 67–76.

[eQu15] eQualit.ie. *(N+1)SEC*. 2015. URL: learn.equalit.ie/wiki/Np1sec (visited on 2015-02-28).

[Fac14] Facebook. *Facebook Help Center*. 2014. URL: https://www.facebook.com/help/ (visited on 2014-11-03).

[FHM+12] Sascha Fahl, Marian Harbach, Thomas Muders, Matthew Smith, and Uwe Sander. "Helping Johnny 2.0 to Encrypt His Facebook Conversations". In: *Symposium on Usable Privacy and Security*. ACM. 2012.

[Fis05] Marc Fischlin. "Completely Non-Malleable Schemes". In: *Automata, Languages and Programming*. Springer, 2005, pp. 779–790.

[FL04] Denis Fomin and Yann Leboulanger. *Gajim, a Jabber/XMPP client*. 2004. URL: https://gajim.org/ (visited on 2014-11-02).

[FLK+13] Michael Farb, Yue-Hsun Lin, Tiffany Hyun-Jin Kim, Jonathan McCune, and Adrian Perrig. "SafeSlinger: Easy-to-Use and Secure Public-Key Exchange". In: *International Conference on Mobile Computing & Networking*. ACM. 2013, pp. 417–428.

[FMB+14] Tilman Frosch, Christian Mainka, Christoph Bader, Florian Bergsma, Joerg Schwenk, and Thorsten Holz. *How Secure is TextSecure?* Cryptology ePrint Archive Report 2014/904. 2014. URL: https://eprint.iacr.org/2014/904.

[FN94] Amos Fiat and Moni Naor. "Broadcast Encryption". In: *Advances in Cryptology–CRYPTO'93*. Springer. 1994, pp. 480–491.

[FS87] Amos Fiat and Adi Shamir. "How To Prove Yourself: Practical Solutions to Identification and Signature Problems". In: *Advances in Cryptology–CRYPTO'86*. Springer. 1987, pp. 186–194.

[GIJ+12] Martin Georgiev, Subodh Iyengar, Suman Jana, Rishita Anubhai, Dan Boneh, and Vitaly Shmatikov. "The Most Dangerous Code in the World: Validating SSL Certificates in Non-Browser Software". In: *Conference on Computer and Communications Security*. ACM. 2012, pp. 38–49.

[GJ04] Philippe Golle and Ari Juels. "Dining Cryptographers Revisited". In: *Advances in Cryptology–EUROCRYPT*. Springer, 2004, pp. 456–473.

[GM05] Simson L Garfinkel and Robert C Miller. "Johnny 2: A User Test of Key Continuity Management with S/MIME and Outlook Express". In: *Symposium on Usable Privacy and Security*. ACM. 2005, pp. 13–24.

[GM15] Matthew Green and Ian Miers. "Forward Secure Asynchronous Messaging from Puncturable Encryption". In: *Symposium on Security and Privacy*. IEEE. 2015.

[GMS+05] Simson L Garfinkel, David Margrave, Jeffrey I Schiller, Erik Nordlander, and Robert C Miller. "How to Make Secure Email Easier To Use". In: *SIGCHI Conference on Human Factors in Computing Systems*. ACM. 2005, pp. 701–710.

[Go 09] Go Project. *The Go Programming Language*. 2009. URL: https://golang.org/ (visited on 2015-04-13).

[Gol14] GoldBug Project. *GoldBug - Secure Instant Messenger*. 2014. URL: http://goldbug.sourceforge.net/ (visited on 2014-11-02).

[Goo14a] Google. *End-To-End*. 2014. URL: https://github.com/google/end-to-end (visited on 2015-02-22).

[Goo14b] Google. *Google Hangouts - Video Conferencing & Meeting for Business*. 2014. URL: https://www.google.com/work/apps/business/products/hangouts/ (visited on 2014-11-03).

[GRPS03] Sharad Goel, Mark Robson, Milo Polte, and Emin Sirer. *Herbivore: A Scalable and Efficient Protocol for Anonymous Communication*. Tech. rep. TR2003-1890. Cornell University, 2003.

[GS07] Prateek Gupta and Vitaly Shmatikov. "Security Analysis of Voice-over-IP Protocols". In: *Computer Security Foundations Symposium*. IEEE. 2007, pp. 49–63.

[GUVC09] Ian Goldberg, Berkant Ustaoğlu, Matthew D Van Gundy, and Hao Chen. "Multi-party Off-the-Record Messaging". In: *Conference on Computer and Communications Security*. ACM. 2009, pp. 358–368.

[Hea12] Christopher C. D. Head. "Anonycaster: Simple, Efficient Anonymous Group Communication". Available from https://blogs.ubc.ca/computersecurity/files/2012/04/anonycaster.pdf. 2012. (Visited on 2014-11-02).

[Hea14] Mike Hearn. *Value of deniability*. Mailing list discussion. 2014. URL: https://moderncrypto.org/mail-archive/messaging/2014/001173.html (visited on 2015-04-02).

[JM95] Bonnie E John and Matthew M Mashyna. *Evaluating a Multimedia Authoring Tool with Cognitive Walkthrough and Think-Aloud User Studies*. Tech. rep. DTIC, 1995.

[JS08] Shaoquan Jiang and Reihaneh Safavi-Naini. "An Efficient Fully Deniable Key Exchange Protocol". In: *Financial Cryptography and Data Security*. Springer, 2008.

[JY96] Markus Jakobsson and Moti Yung. "Proving Without Knowing: On Oblivious, Agnostic and Blindfolded Provers". In: *Advances in Cryptology–CRYPTO*. Springer. 1996, pp. 186–200.

[Kat03] Jonathan Katz. "Efficient and Non-Malleable Proofs of Plaintext Knowledge and Applications". In: *Advances in Cryptology–EUROCRYPT*. Springer, 2003, pp. 211–228.

[Key14] Keybase. *Keybase*. 2014. URL: https://keybase.io/ (visited on 2015-05-14).

[KFR09] Ronald Kainda, Ivan Flechais, and A. W. Roscoe. "Usability and Security of Out-Of-Band Channels in Secure Device Pairing Protocols". In: *Symposium on Usable Privacy and Security*. ACM, 2009.

[KP05] Caroline Kudla and Kenneth G Paterson. "Modular Security Proofs for Key Agreement Protocols". In: *Advances in Cryptology–ASIACRYPT*. Springer, 2005, pp. 549–565.

[Kra03] Hugo Krawczyk. "SIGMA: The 'SIGn-and-MAc' Approach to Authenticated Diffie-Hellman and its Use in the IKE protocols". In: *Advances in Cryptology–CRYPTO 2003*. Springer, 2003, pp. 400–425.

[Kra05] Hugo Krawczyk. "HMQV: A High-Performance Secure Diffie-Hellman Protocol". In: *Advances in Cryptology–CRYPTO*. Springer. 2005, pp. 546–566.

[Kra96] Hugo Krawczyk. "SKEME: A Versatile Secure Key Exchange Mechanism for Internet". In: *Network and Distributed System Security Symposium*. IEEE. 1996, pp. 114–127.

[KT08] Apu Kapadia and Nikos Triandopoulos. "Halo: High-Assurance Locate for Distributed Hash Tables". In: *Network and Distributed System Security Symposium*. Internet Society, 2008.

[KTN04] Hiroaki Kikuchi, Minako Tada, and Shohachiro Nakanishi. "Secure Instant Messaging Protocol Preserving Confidentiality against Administrator". In: *International Conference on Advanced Information Networking and Applications*. IEEE. 2004, pp. 27–30.

[Lam79] Leslie Lamport. *Constructing Digital Signatures from a One Way Function*. Tech. rep. CSL-98. SRI International Palo Alto, 1979.

[Lan13a] Adam Langley. *Enhancing digital certificate security*. 2013. URL: http://googleonlinesecurity.blogspot.de/2013/01/enhancing-digital-certificate-security.html (visited on 2014-11-02).

[Lan13b] Adam Langley. *Pond*. 2013. URL: https://pond.imperialviolet.org/ (visited on 2014-11-02).

[LLK13] Ben Laurie, Adam Langley, and Emilia Kasper. *Certificate Transparency*. RFC 6962 (Experimental). Internet Engineering Task Force, 2013. URL: http://tools.ietf.org/rfc/rfc6962.txt.

[LLM07] Brian LaMacchia, Kristin Lauter, and Anton Mityagin. "Stronger Security of Authenticated Key Exchange". In: *Provable Security*. Springer, 2007, pp. 1–16.

[LM06] Kristin Lauter and Anton Mityagin. "Security Analysis of KEA Authenticated Key Exchange Protocol". In: *Public Key Cryptography – PKC 2006*. Springer, 2006, pp. 378–394.

[LMQ+03] Laurie Law, Alfred Menezes, Minghua Qu, Jerry Solinas, and Scott Vanstone. "An Efficient Protocol for Authenticated Key Agreement". In: *Designs, Codes and Cryptography* 28.2 (2003), pp. 119–134.

[LV09] Benoît Libert and Damien Vergnaud. "Adaptive-ID Secure Revocable Identity-Based Encryption". In: *Topics in Cryptology–CT-RSA*. Springer, 2009, pp. 1–15.

[LVH13] Hong Liu, Eugene Y Vasserman, and Nicholas Hopper. "Improved Group Off-the-Record Messaging". In: *Workshop on Privacy in the Electronic Society*. ACM. 2013, pp. 249–254.

[LY09] Chia-Pei Lee and Chung-Huang Yang. "Design and Implement of a Secure Instant Messaging Service with IC Card". Available from http://crypto.nknu.edu.tw/psnl/publications/2009CPU_SIMICCard.pdf. 2009.

[Lyn06] Ben Lynn. *The Pairing-Based Cryptography Library*. 2006. URL: https://crypto.stanford.edu/pbc/ (visited on 2015-04-13).

[Mad14] Marry Madden. *Public Perceptions of Privacy and Security in the Post-Snowden Era*. 2014. URL: http://www.pewinternet.org/2014/11/12/public-privacy-perceptions/ (visited on 2014-11-14).

[Mar09] Moxie Marlinspike. "More tricks for defeating SSL in practice". In: *Black Hat USA*. 2009.

[MBB+14] Marcela S. Melara, Aaron Blankstein, Joseph Bonneau, Michael J. Freedman, and Edward W. Felten. *CONIKS: A Privacy-Preserving Consistent Key Service for Secure End-to-End Communication*. Cryptology ePrint Archive Report 2014/1004. 2014. URL: https://eprint.iacr.org/2014/1004.

[MBZ12] Vinnie Moscaritolo, Gary Belvin, and Phil Zimmermann. *Silent Circle Instant Messaging Protocol Protocol Specification*. Silent Circle, 2012.

[MD05] Steven J Murdoch and George Danezis. "Low-Cost Traffic Analysis of Tor". In: *Symposium on Security and Privacy*. IEEE. 2005, pp. 183–195.

[Mic14] Microsoft. *Does Skype use encryption?* 2014. URL: https://support.skype.com/en/faq/FA31/does-skype-use-encryption (visited on 2014-11-02).

[MM02] Petar Maymounkov and David Mazières. “Kademlia: A Peer-to-Peer Information System Based on the XOR Metric”. In: *Peer-to-Peer Systems*. Springer, 2002, pp. 53–65.

[MQV95] Alfred J Menezes, Minghua Qu, and Scott A Vanstone. “Some New Key Agreement Protocols Providing Implicit Authentication”. In: *Selected Areas in Cryptography*. 1995, pp. 22–32.

[Mv06] Mohammad Mannan and Paul C van Oorschot. “A Protocol for Secure Public Instant Messaging”. In: *Financial Cryptography and Data Security*. Springer, 2006, pp. 20–35.

[Nak08] Satoshi Nakamoto. “Bitcoin: A Peer-to-Peer Electronic Cash System”. Self-published. 2008.

[Nam11] Namecoin Project. *Namecoin*. 2011. URL: https://namecoin.info/ (visited on 2014-11-02).

[Nao02] Moni Naor. “Deniable Ring Authentication”. In: *Advances in Cryptology–CRYPTO 2002*. Springer, 2002, pp. 481–498.

[Nat98] National Security Agency. *SKIPJACK and KEA Algorithm Specifications*. Version Version 2.0. 1998. URL: http://csrc.nist.gov/groups/ST/toolkit/documents/skipjack/skipjack.pdf (visited on 2014-11-02).

[Nie92] Jakob Nielsen. “Finding Usability Problems Through Heuristic Evaluation”. In: *SIGCHI Conference on Human Factors in Computing Systems*. ACM. 1992, pp. 373–380.

[Nie94] Jakob Nielsen. “Usability inspection methods”. In: *Conference Companion on Human Factors in Computing Systems*. ACM. 1994, pp. 413–414.

[Ope13a] Open Whisper Systems. *Advanced cryptographic ratcheting*. 2013. URL: https://www.whispersystems.org/blog/advanced-ratcheting/ (visited on 2014-11-02).

[Ope13b] Open Whisper Systems. *Forward Secrecy for Asynchronous Messages*. 2013. URL: https://whispersystems.org/blog/asynchronous-security/ (visited on 2015-05-14).

[Ope13c] Open Whisper Systems. *Open WhisperSystems*. 2013. URL: https://www.whispersystems.org/ (visited on 2014-11-02).

[Ope13d] Open Whisper Systems. *Simplifying OTR deniability*. 2013. URL: https://www.whispersystems.org/blog/simplifying-otr-deniability (visited on 2014-11-02).

[Ope14a] Open Whisper Systems. *Open Whisper Systems partners with WhatsApp to provide end-to-end encryption*. 2014. URL: https://www.whispersystems.org/blog/whatsapp/ (visited on 2014-12-23).

[Ope14b] Open Whisper Systems. *Private Group Messaging*. 2014. URL: https://www.whispersystems.org/blog/private-groups/ (visited on 2014-11-02).

[Pas03] Rafael Pass. "On Deniability in the Common Reference String and Random Oracle Model". In: *Advances in Cryptology–CRYPTO 2003*. Springer, 2003, pp. 316–337.

[Per13] Trevor Perrin. *Axolotl Ratchet*. 2013. URL: https://github.com/trevp/axolotl/wiki (visited on 2014-11-02).

[PGES05] Johan A Pouwelse, Paweł Garbacki, Dick Epema, and Henk Sips. "The BitTorrent P2P File-Sharing System: Measurements and Analysis". In: *Peer-to-Peer Systems IV*. Springer, 2005, pp. 205–216.

[PHJ+08] Martin Petraschek, Thomas Hoeher, Oliver Jung, Helmut Hlavacs, and Wilfried N Gansterer. "Security and Usability Aspects of Man-in-the-Middle Attacks on ZRTP". In: *Journal of Universal Computer Science* 14.5 (2008), pp. 673–692.

[PNZE11] Andriy Panchenko, Lukas Niessen, Andreas Zinnen, and Thomas Engel. "Website Fingerprinting in Onion Routing Based Anonymization Networks". In: *Workshop on Privacy in the Electronic Society*. ACM. 2011, pp. 103–114.

[Ric14] Ricochet Project. *Anonymous and serverless instant messaging that just works*. 2014. URL: https://github.com/ricochet-im/ricochet (visited on 2015-04-01).

[RKAG07] Joel Reardon, Alan Kligman, Brian Agala, and Ian Goldberg. *KleeQ: Asynchronous Key Management for Dynamic Ad-Hoc Networks*. Tech. rep. University of Waterloo, 2007.

[RKB+13] Scott Ruoti, Nathan Kim, Ben Burgon, Timothy van der Horst, and Kent Seamons. "Confused Johnny: When Automatic Encryption Leads to Confusion and Mistakes". In: *Symposium on Usable Privacy and Security*. ACM. 2013.

[RL96] Ronald L Rivest and Butler Lampson. "SDSI – A Simple Distributed Security Infrastructure". Manuscript. 1996.

[RR02] Leonid Reyzin and Natan Reyzin. "Better than BiBa: Short One-time Signatures with Fast Signing and Verifying". In: *Information Security and Privacy*. Springer. 2002, pp. 144–153.

[RS92] Charles Rackoff and Daniel R Simon. "Non-Interactive Zero-Knowledge Proof of Knowledge and Chosen Ciphertext Attack". In: *Advances in Cryptology–CRYPTO'91*. Springer. 1992, pp. 433–444.

[RSG98] Michael G Reed, Paul F Syverson, and David M Goldschlag. “Anonymous Connections and Onion Routing”. In: *Selected Areas in Communications* 16.4 (1998), pp. 482–494.

[RST01] Ronald L Rivest, Adi Shamir, and Yael Tauman. “How to Leak a Secret”. In: *Advances in Cryptology–ASIACRYPT 2001*. Springer, 2001, pp. 552–565.

[RT10] Blake Ramsdell and Sean Turner. *Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.2 Message Specification*. RFC 5751 (Proposed Standard). Internet Engineering Task Force, 2010. URL: http://tools.ietf.org/rfc/rfc5751.txt.

[RVR14] Karen Renaud, Melanie Volkamer, and Arne Renkema-Padmos. “Why Doesn’t Jane Protect Her Privacy?” In: *Privacy Enhancing Technologies*. Springer. 2014, pp. 244–262.

[Rya14] Mark D Ryan. “Enhanced Certificate Transparency and End-to-end Encrypted Mail”. In: *Network and Distributed System Security Symposium*. Internet Society, 2014.

[Sai11] Peter Saint-Andre. *Extensible Messaging and Presence Protocol (XMPP): Core*. RFC 6120 (Proposed Standard). Internet Engineering Task Force, 2011. URL: http://tools.ietf.org/rfc/rfc6120.txt.

[SBKH06] Steve Sheng, Levi Broderick, Colleen Alison Koranda, and Jeremy J Hyland. “Why Johnny Still Can’t Encrypt: Evaluating the Usability of Email Encryption Software”. In: *Symposium On Usable Privacy and Security*. ACM. 2006.

[Sch91] Claus-Peter Schnorr. “Efficient Signature Generation by Smart Cards”. In: *Journal of Cryptology* 4.3 (1991), pp. 161–174.

[SCM05] Len Sassaman, Bram Cohen, and Nick Mathewson. “The Pynchon Gate: A Secure Method of Pseudonymous Mail Retrieval”. In: *Workshop on Privacy in the Electronic Society*. ACM. 2005, pp. 1–9.

[SCW+14] Ewa Syta, Henry Corrigan-Gibbs, Shu-Chun Weng, David Wolinsky, Bryan Ford, and Aaron Johnson. “Security Analysis of Accountable Anonymity in Dissent”. In: *Transactions on Information and System Security* 17.1 (2014), p. 4.

[SDOF07] Stuart E Schechter, Rachna Dhamija, Andy Ozment, and Ian Fischer. “The Emperor’s New Security Indicators: An evaluation of website authentication and the effect of role playing on usability studies”. In: *Symposium on Security and Privacy*. IEEE. 2007, pp. 51–65.

[SEA+09] Joshua Sunshine, Serge Egelman, Hazim Almuhimedi, Neha Atri, and Lorrie Faith Cranor. “Crying Wolf: An Empirical Study of SSL Warning Effectiveness”. In: *Security Symposium*. USENIX, 2009, pp. 399–416.

[SFK+12] Sebastian Schrittwieser, Peter Frühwirt, Peter Kieseberg, Manuel Leithner, Martin Mulazzani, Markus Huber, and Edgar R Weippl. "Guess Who's Texting You? Evaluating the Security of Smartphone Messaging Applications". In: *Network and Distributed System Security Symposium*. Internet Society, 2012.

[Sha85] Adi Shamir. "Identity-Based Cryptosystems and Signature Schemes". In: *Advances in Cryptology*. Springer. 1985, pp. 47–53.

[Shi00] Robert Shirey. *Internet Security Glossary*. RFC 2828 (Informational). Obsoleted by RFC 4949. Internet Engineering Task Force, 2000. URL: http://tools.ietf.org/rfc/rfc2828.txt.

[SIL00] SILC Project. *SILC – Secure Internet Live Conferencing*. 2000. URL: http://silcnet.org/ (visited on 2014-11-02).

[SML+01] Ion Stoica, Robert Morris, David Liben-Nowell, David Karger, M Frans Kaashoek, Frank Dabek, and Hari Balakrishnan. "Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications". In: *SIGCOMM Computer Communication Review* 31.4 (2001), pp. 149–160.

[SS14] Maliheh Shirvanian and Nitesh Saxena. "Wiretapping via Mimicry: Short Voice Imitation Man-in-the-Middle Attacks on Crypto Phones". In: *Conference on Computer and Communications Security*. ACM. 2014, pp. 868–879.

[SW05] Amit Sahai and Brent Waters. "Fuzzy Identity-Based Encryption". In: *Advances in Cryptology–EUROCRYPT*. Springer, 2005, pp. 457–473.

[SW07] Hovav Shacham and Brent Waters. "Efficient Ring Signatures without Random Oracles". In: *Public Key Cryptography*. Springer, 2007, pp. 166–180.

[SYG08] Ryan Stedman, Kayo Yoshida, and Ian Goldberg. "A User Study of Off-the-Record Messaging". In: *Symposium on Usable Privacy and Security*. ACM. 2008, pp. 95–104.

[Tel14] Telegram. *Telegram Messenger*. 2014. URL: https://telegram.org/ (visited on 2014-11-02).

[TZ05] Amandeep Thukral and Xukai Zou. "Secure Group Instant Messaging Using Cryptographic Primitives". In: *Networking and Mobile Computing*. Springer, 2005, pp. 1002–1011.

[UDB+15a] Nik Unger, Sergej Dechand, Joseph Bonneau, Sascha Fahl, Henning Perl, Ian Goldberg, and Matthew Smith. "SoK: Secure Messaging". In: *Symposium on Security and Privacy*. IEEE. 2015.

[UDB+15b] Nik Unger, Sergej Dechand, Joseph Bonneau, Sascha Fahl, Henning Perl, Ian Goldberg, and Matthew Smith. *SoK: Secure Messaging*. Tech. rep. 2015-02. CACR, 2015. URL: http://cacr.uwaterloo.ca/techreports/2015/cacr2015-02.pdf.

[UHHC11] Alexander Ulrich, Ralph Holz, Peter Hauck, and Georg Carle. "Investigating the OpenPGP Web of Trust". In: *Computer Security–ESORICS*. Springer, 2011, pp. 489–507.

[Van13] Matthew Van Gundy. "Improved Deniable Signature Key Exchange for mpOTR". Available from http://matt.singlethink.net/projects/mpotr/improved-dske.pdf. 2013.

[VAS11] VASCO. *DigiNotar reports security incident*. 2011. URL: https://www.vasco.com/company/about_vasco/press_room/news_archive/2011/news_diginotar_reports_security_incident.aspx (visited on 2014-11-03).

[VC12] Matthew D Van Gundy and Hao Chen. "OldBlue: Causal Broadcast In A Mutually Suspicious Environment (Working Draft)". Available from http://matt.singlethink.net/projects/mpotr/oldblue-draft.pdf. 2012.

[VV08] Carmine Ventre and Ivan Visconti. "Completely Non-Malleable Encryption Revisited". In: *Public Key Cryptography–PKC 2008*. Springer, 2008, pp. 65–84.

[Wal08] Shabsi Walfish. "Enhanced Security Models for Network Protocols". PhD thesis. New York University, 2008.

[WAP08] Dan Wendlandt, David G Andersen, and Adrian Perrig. "Perspectives: Improving SSH-style Host Authentication with Multi-Path Probing". In: *Annual Technical Conference*. USENIX, 2008, pp. 321–334.

[War14] Brian Warner. *Pairing Problems*. 2014. URL: https://blog.mozilla.org/warner/2014/04/02/pairing-problems/ (visited on 2014-04-02).

[WB12] Qiyan Wang and Nikita Borisov. "Octopus: A Secure and Anonymous DHT Lookup". In: *International Conference on Distributed Computing Systems*. IEEE. 2012, pp. 325–334.

[WCFJ12] David Isaac Wolinsky, Henry Corrigan-Gibbs, Bryan Ford, and Aaron Johnson. "Dissent in Numbers: Making Strong Anonymity Scale". In: *Conference on Operating Systems Design and Implementation*. USENIX, 2012, pp. 179–182.

[Wic14] Wickr. *Wickr – Top Secret Messenger*. 2014. URL: https://wickr.com/ (visited on 2014-11-02).

[WLL13] Chang-Ji Wang, Wen-Long Lin, and Hai-Tao Lin. "Design of An Instant Messaging System Using Identity Based Cryptosystems". In: *International Conference on Emerging Intelligent Data and Web Technologies*. IEEE. 2013, pp. 277–281.

[WLXZ14] Changji Wang, Yuan Li, Xiaonan Xia, and Kangjia Zheng. "An Efficient and Provable Secure Revocable Identity-Based Encryption Scheme". In: *PLOS ONE* (2014).

[WP89] Michael Waidner and Birgit Pfitzmann. “The Dining Cryptographers in the Disco: Unconditional Sender and Recipient Untraceability with Computationally Secure Serviceability”. In: *Advances in Cryptology–EUROCRYPT*. Springer, 1989.

[WS09] Bin Wang and ZhaoXia Song. “A non-interactive deniable authentication scheme based on designated verifier proofs”. In: *Information Sciences* 179.6 (2009), pp. 858–865.

[WSG14a] Matthias Wachs, Martin Schanzenbach, and Christian Grothoff. “A Censorship-Resistant, Privacy-Enhancing and Fully Decentralized Name System”. In: *Cryptology and Network Security*. Springer, 2014, pp. 127–142.

[WSG14b] Matthias Wachs, Martin Schanzenbach, and Christian Grothoff. “On the Feasibility of a Censorship Resistant Decentralized Name System”. In: *Foundations and Practice of Security*. Springer, 2014, pp. 19–30.

[WT99] Alma Whitten and J Doug Tygar. “Why Johnny Can’t Encrypt: A Usability Evaluation of PGP 5.0”. In: *Security Symposium*. USENIX, 1999.

[WWX14] Weiqiang Wen, Libin Wang, and Min Xie. *One-Round Deniable Key Exchange with Perfect Forward Security*. Tech. rep. 2014/904. Cryptology ePrint Archive, 2014. URL: https://eprint.iacr.org/2014/661.

[YK07] Chung-Huang Yang and Tzong-Yih Kuo. “The Design and Implementation of a Secure Instant Messaging Key Exchange Protocol”. Available from http://crypto.nknu.edu.tw/psnl/publications/2007Technology_SIMPP.pdf. 2007.

[YKAL08] Chung-Huang Yang, Tzong-Yih Kuo, TaeNam Ahn, and Chia-Pei Lee. “Design and Implementation of a Secure Instant Messaging Service based on Elliptic-Curve Cryptography”. In: *Journal of Computers* 18.4 (2008), pp. 31–38.

[YLP11] Taek-Young Youn, Changhoon Lee, and Young-Ho Park. “An efficient non-interactive deniable authentication scheme based on trapdoor commitment schemes”. In: *Computer Communications* 34.3 (2011), pp. 353–357.

[YYZZ10] Andrew C. Yao, Frances F. Yao, Yunlei Zhao, and Bin Zhu. “Deniable Internet Key Exchange”. In: *Applied Cryptography and Network Security*. Springer. 2010, pp. 329–348.

[YZ13] Andrew Chi-Chih Yao and Yunlei Zhao. “OAKE: A New Family of Implicitly Authenticated Diffie-Hellman Protocols”. In: *Conference on Computer and Communications Security*. ACM. 2013, pp. 1113–1128.

[ZANS12] Huafei Zhu, Tadashi Araragi, Takashi Nishide, and Kouichi Sakurai. “Universally Composable Non-committing Encryptions in the Presence of Adaptive Adversaries”. In: *e-Business and Telecommunications*. Springer, 2012, pp. 274–288.

[Zim95] Philip R Zimmermann. *The Official PGP User's Guide*. MIT Press Cambridge, 1995.

[ZJC11] Philip Zimmermann, Alan Johnston, and Jon Callas. *ZRTP: Media Path Key Agreement for Unicast Secure RTP*. RFC 6189 (Informational). Internet Engineering Task Force, 2011. URL: http://www.ietf.org/rfc/rfc6189.txt.

[ZMM10] JunWei Zhang, JianFeng Ma, and SangJae Moon. "Universally composable one-time signature and broadcast authentication". In: *Science China Information Sciences* 53.3 (2010), pp. 567–580.