Key establishment — security models, protocols and usage

by

Berkant Ustaoğlu

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Combinatorics and Optimization

Waterloo, Ontario, Canada, 2008

I hereby declare that I am the sole author of this 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.

Abstract

Key establishment is the process whereby two or more parties derive a shared secret, typically used for subsequent confidential communication. However, identifying the exact security requirements for key establishment protocols is a non-trivial task. This thesis compares, extends and merges existing security definitions and models for key establishment protocols.

The primary focus is on two-party key agreement schemes in the public-key setting. On one hand new protocols are proposed and analyzed in the existing Canetti-Krawzcyk model. On the other hand the thesis develops a security model and novel definition that capture the essential security attributes of the standardized Unified Model key agreement protocol. These analyses lead to the development of a new security model and related definitions that combine and extend the Canetti-Krawzcyk pre- and post- specified peer models in terms of provided security assurances.

The thesis also provides a complete analysis of a one-pass key establishment scheme. There are security goals that no one-pass key establishment scheme can achieve, and hence the two-pass security models and definitions need to be adapted for one-pass protocols. The analysis provided here includes the description of the required modification to the underlying security model. Finally, a complete security argument meeting these altered conditions is presented as evidence supporting the security of the one-pass scheme.

Lastly, validation and reusing short lived key pairs are related to efficiency, which is a major objective in practice. The thesis considers the formal implication of omitting validation steps and reusing short lived key pairs. The conclusions reached support the generally accepted cryptographic conventions that incoming messages should not be blindly trusted and extra care should be taken when key pairs are reused.

Acknowledgements

It is my pleasure to say a very special “thank you” to Alfred Menezes. Without your support, guidance, help and encouragement this would have been an impossible task for me. Thank you!

During my degree I spent many enjoyable moments with people from my department to you guys “thanks for all you did for me”. Alison, Aşan, Burkay, Emre Karakoç, Gülay, Irene, Jeff Wong, John Taranu, Koray — you helped me fully live my days, you let me see and try and live things I would have missed without you, for that I thank you once more.

Elodie, the first time you told me “hi” was my luckiest moment in Waterloo. The moment I met the friend who would, from the very beginning of this journey, be always there to help me. Merci beaucoup, mon cher amie!

To CACR members — you were always ready to engage in fruitful discussions. I am grateful for all the comments and helps you gave me. I would also like to thank my committee members for carefully examining my thesis and providing me with their comments and suggestions.

And last but not least I would like to thank NSERC and MITACS for the vital financial support they provided.

Feride ve Mürsel Ustaoğlu
sevgilerimle, sizin için

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

The main objective of cryptography is to provide means for confidential communication [72, §1.1]. In a face-to-face environment, two parties Alice and Bob achieve confidentiality by actively guarding themselves against eavesdroppers. By contrast, over a public communication channel like the Internet, neither Alice nor Bob can prevent eavesdroppers from reading their messages. Therefore, over a public channel confidentiality is harder to achieve.

Symmetric-key cryptography offers means to arrange confidential communication over a public connection. Even though Alice cannot control who is reading her outgoing messages, by encrypting her messages with symmetric-key algorithms she is assured that only Bob can extract the confidential information from her messages. The main idea in symmetric-key cryptography is having Alice and Bob agree on a *shared secret key* κ before the communication takes place. The key κ is subsequently used by Alice and Bob to encrypt and decrypt their messages. The shared key must not be revealed to any other entity Eve. If Eve learns κ, then she is able to extract the secret information from the messages encrypted with κ, in which case Alice and Bob do not have confidentiality.

The *one-time pad* is a classic example of symmetric-key cryptography. The idea, attributed to Gilbert Vernam, was analysed by Shannon [68] who concluded that the one-time pad provides *unconditional security*. However, for security the shared secret key κ should not be reused and κ has to be as long as the communicated message. As a result, the one-time pad is an inefficient way to achieve confidentiality.

Efficiency can be improved at the expense of security assurances. Parties that share a relatively short key κ can use either a *stream* or a *block* cipher to achieve confidential communication. With stream ciphers messages are encrypted by adding them bitwise modulo two with a pseudo-random string that is derived from κ. With block ciphers the message is encrypted one block at a time, where a block is a sub-message of a predefined bit length. A classical example of a stream cipher is RC4, whereas DES is a typical example of a block cipher; see [55, Chapters 6 and 7] for further discussion on stream and block ciphers. Even though stream and block ciphers do not provide unconditional security, they provide an acceptable trade-off between security and efficiency.

If parties possess shared secret keys, symmetric-key ciphers provide efficient means for confidential communication. However managing shared keys is not an easy task. First of all, a large number of potential communicating peers implies at least the same number of shared keys a party must manage. Secondly, in cases where parties are unable to frequently update their shared keys, the possibility of an adversary learning a secret key increases over time. Last but not least, meeting face-to-face to agree on a shared key may not be possible at all, for example when the first contact is made over a public channel but the parties are geographically separated. Hence, shared key management is not a solution that scales well and in cases where parties cannot meet, confidentiality cannot be achieved.

Diffie and Hellman [24] proposed the idea of *public-key cryptography*. Their idea can be used to address the shared key management problem. Unlike symmetric-key cryptography, where the encryption and decryption algorithms are closely related, in public-key cryptography the decryption and encryption algorithms are different. In fact, the ability to encrypt messages, does not necessarily imply the ability to decrypt messages. Rivest, Shamir and Adleman [65] and ElGamal [28] presented examples of public-key cryptosystems. With public-key cryptography Alice and Bob can establish a new shared secret key without meeting in person, and as a result, avoid the problems related to shared key management.

Building a confidential and authentic communication channel between two parties is broadly divided into two stages. At the first stage public-key cryptography is used to obtain an authenticated shared secret key κ. At the second stage κ is used with a symmetric-key cipher to simulate a confidential channel. *Key establishment* is a cryptographic primitive that allows Alice and Bob to establish a shared secret key at the first stage of building the communication channel. The process of key establishment can be divided into two classes: *key transport* and *key agreement*.

In a key transport protocol Alice selects a secret key κ and securely transmits

κ to Bob. A typical use of key transport is encrypted emails. Alice picks κ and encrypts her message m to Bob with κ using symmetric-key techniques. She also encrypts κ using public-key cryptography, so that only Bob can decrypt and obtain κ. Alice then sends the encrypted κ and m to Bob. Bob first decrypts κ and thereafter obtains the rest of Alice's message.

In a key agreement protocol both Alice and Bob contribute to the shared key. This scenario is more common in an environment where both parties actively exchange messages. With key agreement protocols both parties can achieve a wider range of security goals than with key transport protocols.

1.2 Goals of key establishment

The goal of two party key establishment is to provide communicating partners with a shared key κ, that is secret and authentic. In other words, κ should resist derivation attempts from entities other than the two participants and furthermore each participant should be assured that only a specified party could also obtain κ.

In the ephemeral Diffie-Hellman (eDH) [24] key agreement protocol depicted in Figure 1.1, Alice and Bob pick ephemeral (short-term) private keys x and y respectively, and exchange ephemeral public keys $X = g^x$ and $Y = g^y$. Both X and Y belong to a publicly known group $\mathcal{G} = \langle g \rangle$ of prime order q and x, y are elements of the integers modulo q. With the knowledge of Y and x, Alice computes the ephemeral shared secret $\sigma_e = Y^x = g^{xy}$, and similarly, using X and y, Bob computes the same ephemeral shared secret $\sigma_e = X^y = g^{xy}$. Thereafter, Alice and Bob derive the shared secret key κ using a suitable key derivation function $\mathcal{H}$.

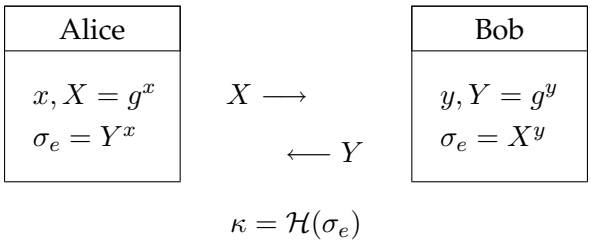


Figure 1.1: Ephemeral Diffie-Hellman

An adversary that only observes the exchanged messages is said to be *passive*. An adversary who modifies, replays, injects or reroutes messages is called *active*. Under the CDH assumption (see §1.5) the eDH protocol is secure against passive adversaries, but insecure against active adversaries. In particular, eDH succumbs

to the well known *person-in-the-middle* attack, where an active adversary Oscar substitutes X and Y with an ephemeral public key $Z = g^z$ of Oscar's choice. As a result, Alice derives a session key $\kappa_x = \mathcal{H}(Z^x)$ and Bob derives a session key $\kappa_y = \mathcal{H}(Z^y)$. Oscar can derive both $\kappa_x = \mathcal{H}(X^z)$ and $\kappa_y = \mathcal{H}(Y^z)$, and subsequently he can read and control the information exchanged between Alice and Bob. The problem with eDH is lack of *authentication* of the exchanged ephemeral public keys.

Digital signatures are public-key algorithms that can be used to authenticate data. By using digital signatures, *trusted third parties* can issue *certificates* that bind public information to Alice's identity. Typically certificates bind a *static* (long-term) public key $A = g^a$ to Alice, where a is Alice's static private key. Similarly, Bob can obtain a certificate for his static public key $B = g^b$. Any party that has an authentic copy of the trusted third party's public key can verify that a certificate is authentic.

In the static Diffie-Hellman (sDH) key agreement protocol Alice and Bob exchange static public keys and certificates instead of ephemeral public keys. As in the ephemeral Diffie-Hellman protocol, Alice and Bob compute the same shared secret $\sigma_s = g^{ab}$ and thereafter obtain a session key $\kappa = \mathcal{H}(\sigma_s)$. Unlike in the ephemeral Diffie-Hellman protocol, where Alice has no assurance that the incoming public key Y was selected by Bob, in the static Diffie-Hellman protocol Alice is assured via certificate verification that the public key B belongs to Bob.

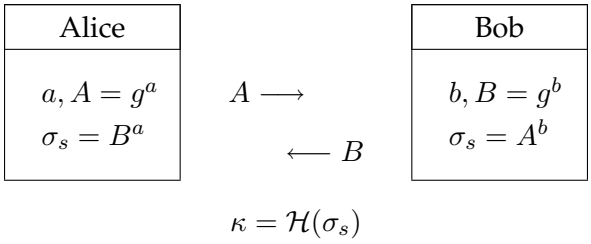


Figure 1.2: Static Diffie-Hellman

A *session* is an execution of the key establishment protocol. *Known key security* (KKS) considers the security of a session key in the event that session keys of other sessions were compromised. Session keys agreed between Alice and Bob may be used in different applications. These applications may have different levels of security and an application with a low security level may leak a particular session key κ. The problem with sDH is that the session key does not change, and therefore the leakage of κ compromises all protocol runs between Alice and Bob.

To prevent KKS attacks Alice needs a fresh session key for each protocol run.

Consider augmenting the static Diffie-Hellman protocol by adding *nonces* (freshly generated ephemeral values) to the key derivation function to obtain the nonce Diffie-Hellman (nDH) key agreement protocol depicted in Figure 1.3. The nonce N_A, generated by Alice, assures Alice that the session keys she computes are independent from each other. Bob obtains similar assurances via N_B. For alternatives to nonces see [16, §1.5].

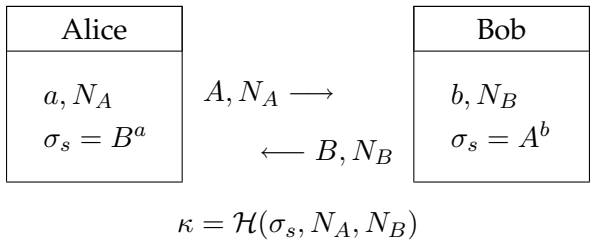


Figure 1.3: Nonce Diffie-Hellman

Forward secrecy (FS) considers the security of previously established session keys in the event that static private keys of communicating partners are compromised. FS is important in scenarios where the secrecy of the information exchanged between Alice and Bob has to be maintained for a long period of time, during which the probability that the adversary learns Alice's or Bob's static private key may increase. In nDH an adversary Oscar may record nonces N_A and N_B. In the future if Oscar learns the static private key of either Alice or Bob, Oscar could decrypt all communication encrypted with $\kappa = \mathcal{H}(\sigma_s, N_A, N_B)$. Therefore, nDH does not provide forward secrecy.

In the eDH protocol Alice and Bob do not have static key pairs, hence forward secrecy is trivially provided. Furthermore, session keys from different sessions are independent from each other, providing resilience to KKS attacks. To prevent the person-in-the-middle attack against eDH, Alice and Bob could authenticate their ephemeral public keys using digital signatures. The resulting protocol, called the signed Diffie-Hellman (signedDH) key agreement protocol, is depicted in Figure 1.4.

Typically, a static private key is chosen before any protocol run. Hence Alice can invest sufficient resources to ensure that her static key pair is cryptographically strong. On the other hand, ephemeral keys are often chosen during the protocol run. In some situations Alice may lack access to a sufficiently strong source of randomness and as a result an adversary, Oscar, may be able to guess or compute the private key x corresponding to Alice's ephemeral public key X. By recording the signature $sig_A(X)$ Oscar could impersonate Alice to Bob by replaying the

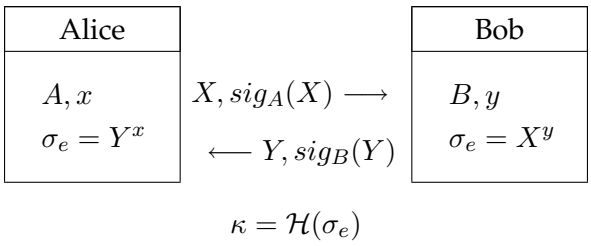


Figure 1.4: Signed Diffie-Hellman

message $X, sig_A(X)$ in the future. Therefore, signedDH is not resilient to leakage of ephemeral private keys (LEP) and the compromise of Alice's ephemeral private key used in one session compromises future protocol runs with Bob.

Instead of using signatures to authenticate ephemeral public keys, Alice and Bob could use both σ_e and σ_s from eDH and sDH, respectively, to derive a session key. The resulting protocol depicted in Figure 1.5 is called the basic Unified Model (basicUM) protocol (see also [13]). The basicUM protocol can be specialized to both eDH and sDH. For example, if static keys are used instead of ephemeral keys, then the basicUM protocol is the sDH protocol. Hence basicUM achieves the same security goals as eDH and as sDH. Furthermore, basicUM resists the above mentioned LEP attack against signedDH.

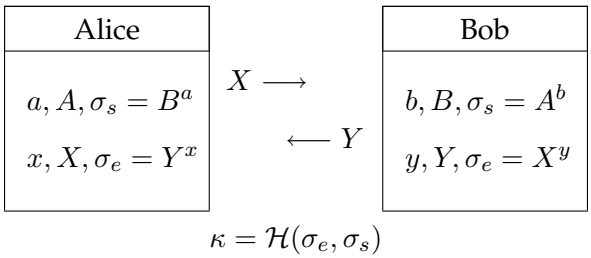


Figure 1.5: Basic Unified Model Protocol

Key compromise impersonation (KCI) resilience [38] refers to the ability of an adversary Oscar who learns Alice's static private key a to impersonate other parties to Alice. If Oscar knows a, then he can certainly masquerade as Alice to Bob. However, in some cases, using a Oscar's goal could be impersonating Bob to Alice. In the basicUM protocol with the knowledge of a Oscar can compute $\sigma_s = B^a = g^{ab}$. By selecting an ephemeral key pair (y, Y) on behalf of Bob, Oscar can compute $\sigma_e = X^y = g^{xy}$. Consequently, the basicUM protocol is not KCI resilient.

The KCI attack against the basicUM protocol assumes that Oscar knows Alice's

ephemeral public key X. Alice could blind her ephemeral public key X to prevent the KCI attack. Matsumoto, Takashima and Imai [50] proposed a class of protocols, now known as the MTI protocols, based on the idea of blinding ephemeral public keys. In the basic MTI protocol in Figure 1.6, Alice sends $\overline{X} = B^x$ to Bob instead of $X = g^x$. Bob responds by sending $\overline{Y} = A^y$. Alice computes the ephemeral shared secret $\sigma_e = g^{xy}$ by computing $\overline{Y}^{a^{-1}x}$. Similarly, Bob computes $\overline{X}^{b^{-1}y}$ to obtain σ_e. Thereafter both Alice and Bob derive the session key $\kappa = \mathcal{H}(\sigma_e)$. The KCI attack against the basicUM protocol cannot be launched against the basic MTI protocol.

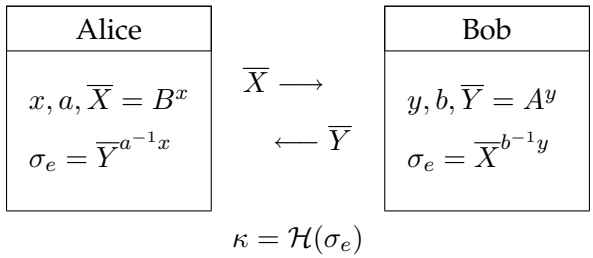


Figure 1.6: Basic MTI protocol

Unknown key share (UKS) attacks [25], sometimes called *identity misbinding* attacks, refer to the ability of an adversary Oscar to coerce Bob into sharing a session key with Alice without Bob's knowledge. In other words Alice and Bob compute the same session key, but while Alice correctly believes she is talking to Bob, Bob thinks he is talking to Oscar. Consider the basic MTI protocol and assume that the certification authority does not require a party to prove possession of the private key that corresponds to the public key the party certifies. In that case Oscar can obtain a certificate binding the static public key $\overline{E} = A^e$ to Oscar; as usual A is Alice's static public key. Suppose Alice starts a session with Bob by sending $\overline{X} = B^x$ to Bob. Oscar could intercept her message and forward $\overline{X}$ to Bob pretending the sender is Oscar. In accordance with the basic MTI protocol, Bob responds to Oscar with outgoing ephemeral public key $\tilde{Y} = \overline{E}^y$. After receiving Bob's response, Oscar computes $\overline{Y} = \tilde{Y}^{e^{-1}} = g^{ay}$ and sends $\overline{Y}$ to Alice. As a result both Alice and Bob compute the same shared secret $\sigma_e = g^{xy}$ (which Oscar cannot compute) and hence derive the same session key $\kappa = \mathcal{H}(\sigma_e)$. But while Alice correctly believes she shares a key with Bob, Bob incorrectly thinks he shares κ with Oscar. Hence Oscar has successfully launched a UKS attack against the basic MTI protocol.

There are many examples of protocols that fall to UKS attacks, but in each case including the identities of the communicating parties in the key derivation prevents the attacks. Even though not a necessary condition, including the identities in the key derivation function appears to be a generic measure that prevents UKS

attacks without affecting other security properties of the protocol under consideration. Naturally, one asks whether there are other generic measures that are sufficient to prevent certain kinds of attacks without affecting other security goals.

Intuition may suggest that an MTI-like ephemeral public key exchange prevents KCI attacks. Consider a combination of the basicUM and the MTI protocols, depicted in Figure 1.7 where the ephemeral keys are exchanged as in the MTI protocol, but the session key is derived as in the basicUM protocol. Suppose Alice starts two simultaneous sessions S^1 and S^2 with Bob using the combined basicUM/MTI protocol. Suppose also that the adversary Oscar has learned Alice's static private key a. Let $\overline{X}_1$ and $\overline{X}_2$ denote Alice's outgoing ephemeral public keys for sessions S^1 and S^2, respectively. Oscar can intercept both messages and compute $\overline{Y}_1 = \overline{X}_1^a$ and $\overline{Y}_2 = \overline{X}_2^a$. Next Oscar can reply to S^1 with $\overline{Y}_2$ and to S^2 with $\overline{Y}_1$, pretending that the replies originated from Bob. As a result, Oscar succeeds in forcing the sessions S^1 and S^2 within Alice to share the same session key $\kappa = \mathcal{H}(g^{x_1 x_2 b}, g^{ab})$. Afterwards, by obtaining the session key from S^1, Oscar is able to violate the security of session S^2, thus succeeding in a KKS attack.

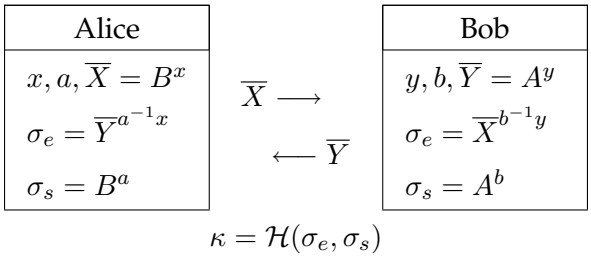


Figure 1.7: Combined basicUM/MTI protocol

Overall, the leakage of private information can have implications for more than one security property. UKS and KKS attacks show that considering only two parties is not sufficient to capture realistic attacks in scenarios where multiple parties can engage in many concurrent sessions with each other. Carefully devised security models can capture multi-party, multi-session environments and at the same time simultaneously cover several security requirements. Models and associated security definitions provide an attractive approach for formal analysis of key establishment security. However, the old fashioned "try to break" approach should not be discarded and furthermore one should be aware that there may be desirable properties of key establishment protocols not captured by the models.

1.3 Number of messages

A single message from Alice to Bob is called a *pass*. The terms *round* and *protocol flow* are used with the same meaning here, even though in the literature the term round describes a message sent by Alice and a response sent by Bob.

In all protocol examples in the previous section Alice and Bob exchange exactly two messages. In practice there are protocols where the number of messages is more than two. Some important scenarios, such as email, require protocols that consist of a single message from Alice to Bob.

The number of protocol passes has implications on the security attributes. For example, no one-pass protocol can provide forward secrecy, whereas two-pass protocols can achieve a weaker variant of forward secrecy, see §3.3 for details. There are protocols with three passes that attain forward secrecy. This thesis focuses on protocols with at most three flows.

1.4 Thesis outline

The next section introduces the notation used in subsequent chapters. Existing models for key establishment protocols are reviewed in Chapter 2. Chapter 3 analyses the one-pass and the two-pass CMQV protocols. Chapter 4 develops a security model that captures security assurances of the Unified Model protocol and provides a security argument for the Unified Model protocol in the developed model. A new 'combined' security model is presented in Chapter 5. Chapter 6 considers some practical aspects of key agreement protocols and lastly, Chapter 7 suggests future research directions.

1.5 Notation, terminology and assumptions

Let q be a prime, and let $\mathbb{Z}_q$ denote the integers modulo q. The symbol $\mathcal{G}$ denotes a multiplicatively-written cyclic group of order q generated by a fixed element g. The set of non-identity elements in $\mathcal{G}$ is denoted by $\mathcal{G}^*$. For group elements $A, B, \ldots$ the corresponding lowercase letters will denote the discrete logarithms in base g; for example $a = \log_g A$, where $a \in \mathbb{Z}_q$. A function $f(n)$ is said to be *negligible* if for every integer d, there is an integer N such that for all integers $n > N$, $f(n) < \frac{1}{n^d}$. Finally, the symbol "$\in_R$" means "is selected uniformly at random from".

Key establishment protocols take place between two parties, from among a set of $\mathbf{n}$ parties, denoted by $\hat{A}$, $\hat{B}$ and so on. Party $\hat{A}$'s static public key is $A \in \mathcal{G}$ and its corresponding static private key is $a = \log_g A$. In general, lower case letters represent secret information, whereas upper case letters are publicly known values. By convention X denotes $\hat{A}$'s ephemeral public key and Y denotes $\hat{B}$'s ephemeral public key.

Introduced by Bellare and Rogaway [10], the *random oracle* is an idealized hash function $\mathcal{H}$ that maps elements from its domain to its image. The function $\mathcal{H}$ is not explicitly described. Instead, there is an oracle that on input x returns the value $y = \mathcal{H}(x)$ and the only way to learn y is to query the oracle with x. As a consequence the result $\mathcal{H}(\tilde{x})$ reveals no information about $\mathcal{H}(x)$ if $x \neq \tilde{x}$. Random oracles are denoted by $\mathcal{H}$, $\mathcal{H}_1$ and $\mathcal{H}_2$.

The discrete logarithm problem (DLP) in $\mathcal{G}$ is defined as follows.

Instance: $X \in \mathcal{G} = \langle g \rangle$.

Solution: $x \in \mathbb{Z}_q$ such that $g^x = X$.

The solution for a DLP instance X is denoted by $\log_g X$.

The computational Diffie-Hellman problem (CDH) in $\mathcal{G}$ is defined as follows.

Instance: $U, V \in \mathcal{G}$.

Solution: $W \in \mathcal{G}$ such that $W = g^{\log_g V \log_g U} = U^{\log_g V} = V^{\log_g U}$.

The solution for a CDH instance (U, V) is denoted by $\mathrm{CDH}(U, V)$.

The decisional Diffie-Hellman problem (DDH) in $\mathcal{G}$ is defined as follows.

Instance: $(U, V, W) \in \mathcal{G}^3$.

Solution: If $\log_g U \times \log_g V = \log_g W \bmod q$, then the result is 1, else the result is 0.

The result of a DDH instance (U, V, W) is denoted by $\mathrm{DDH}(U, V, W)$.

The computational square problem (CSP) in $\mathcal{G}$ is defined as follows.

Instance: $X \in \mathcal{G}$.

Solution: $Y \in \mathcal{G}$ such that $\log_g Y = (\log_g X)^2 \bmod q$.

The solution for a CSP instance X is denoted by $\mathrm{CSP}(X)$.

Pointcheval and Okamoto [60] introduced the computational *gap* problems. The gap Diffie-Hellman problem (GDH) in $\mathcal{G}$ is the CDH problem in $\mathcal{G}$, where the input also includes an oracle that solves the DDH problem in the same group. Similarly, the gap square problem (GSP) in $\mathcal{G}$ is CSP in $\mathcal{G}$, where the input also includes a DDH oracle for $\mathcal{G}$.

In a group $\mathcal{G}$, the CDH assumption holds if there exists no polynomially bounded algorithm that can solve an arbitrary instance of the CDH problem. The GDH, DDH, CSP and GSP assumptions are defined in a similar manner.

A reduction $\mathcal{R}$ from Π to Π' is an algorithm for solving Π that uses as a subroutine a hypothetical oracle for solving Π'. If $\mathcal{R}$ and the Π' oracle are efficient (run in polynomial time), then the reduction is called an efficient (polynomial time) reduction. If there is an efficient (polynomial time) reduction from Π to Π', then Π is said to (polynomial-time) *reduce* to Π'. If Π' also polynomial-time reduces to Π then Π and Π' are said to be *polynomially equivalent*. For example, any (efficient) algorithm that solves the CDH problem also solves the GDH problem, hence the GDH problem reduces to the CDH problem. Similarly, the DDH problem reduces to the CDH problem. The CSP and the CDH problems are known to be polynomially equivalent [51], and it is therefore reasonable to conjecture that the GSP and the GDH problems are also polynomially equivalent.

Chapter 2

Extended Canetti-Krawzcyk model

2.1 Earlier work

The Bellare-Rogaway (BR) [9, 11] and Blake-Wilson, Johnson, Menezes (BJM) [13] models were among the first security models for key establishment protocols. Unlike the BR model, which considers the symmetric-key setting, the BJM model considers the public-key setting. Except for this difference the BJM and the BR model are essentially the same. The remainder of this section gives an outline of the BR model.

The BR model considers a set of entities I, a random shared secret generator, and a security parameter. On input the security parameter the random secret generator outputs shared secrets for every pair in I. A special entity, called the *adversary* who is not in the set I, interacts with a collection of oracles $\Pi^s_{i,j}$, where $i, j \in I$ and s is a natural number. The oracle $\Pi^s_{i,j}$ represents the s'th session that entity i attempts to execute with entity j. The adversary controls all communication and can present an oracle with an incoming message; in return the adversary obtains the oracle response, which can be an outgoing message as described by the protocol, or acceptance or rejection of a session key.

The BR model uses the idea of *matching conversations* to decide which two oracles produce the same session key. A *conversation* for oracle $\Pi^s_{i,j}$ is the ordered concatenation of incoming and outgoing messages. Two oracles $\Pi^s_{i,j}$ and $\Pi^t_{j,i}$ are said to have matching conversations if (i) $\Pi^t_{j,i}$'s outgoing messages constitute $\Pi^s_{i,j}$'s incoming messages; and (ii) $\Pi^s_{i,j}$'s outgoing messages constitute $\Pi^t_{j,i}$'s incoming messages. A protocol *round* is either an incoming or an outgoing message as viewed by an entity. A protocol consisting of at least three rounds is called *secure mutual authentication* protocol if the following conditions hold: (i) if two oracles complete

matching conversations, then they both accept; and (ii) the probability that an oracle accepts without a matching oracle is negligible in the security parameter.

Even though in the BR model the adversary controls the message exchange between oracles, the adversary is not granted access to the session keys accepted by the oracles or to secrets shared between entities. To model leakage of session keys, the adversary is equipped with a *Reveal* query, that takes as input an oracle that accepted a session key κ and returns κ to the adversary. Leaking long-term secret information is modeled by equipping the adversary with a *Corrupt* query that takes as input $i \in I$ and allows the adversary to take over that entity. With this an oracle $\Pi^s_{i,j}$ is said to be *fresh* if (i) the oracle computes and accepts a session key; (ii) the adversary does not issue the *Reveal* query against $\Pi^s_{i,j}$; (iii) the adversary does not issue the *Reveal* query against the oracle matching to $\Pi^s_{i,j}$; and (iv) the adversary does not issue *Corrupt*(i) or *Corrupt*(j).

Upon completing the interaction with the oracles the adversary picks a fresh oracle called the *test* oracle. In response a bit $\gamma \in_R \{0,1\}$ is chosen. If $\gamma = 1$, then adversary is given the session key computed by the test oracle, otherwise a random key. The key establishment protocol is said to be *BR-secure* if (i) in the presence of a *benign* adversary, that is an adversary who faithfully delivers messages, the protocol works as specified; and (ii) no polynomially bounded adversary can guess γ with probability greater than $\frac{1}{2}$ plus a term that is negligible in the security parameter.

2.2 Canetti-Krawzcyk model

The Canetti-Krawzcyk (CK01) model [20] was developed in the public key setting. This section describes the CK01 model and the associated security implications.

2.2.1 CK01 description

An execution of a key establishment protocol is an invocation of a collection of $\mathbf{n}$ *message driven protocols*, where each protocol is initiated within a probabilistic polynomial time machine called a *party*. Parties are *activated* via an *action request* or *incoming message*. Once activated a party either *creates* a separate subroutine within the same party called a *session*, or *updates* an existing session S within the party by submitting the incoming messages to S for further processing. In response to an activation or an incoming message a party creates an outgoing message or another

action request. Key establishment protocols are message driven protocols where sessions play a role similar to the role of oracles in the BR model.

Each party in the CK01 model possesses a certificate that binds its static public keys to the party. A party $\hat{A}$ can be activated to create a session via an action request *Create*$(\hat{A}, \hat{B}, \Psi, role)$ where $\hat{B}$ is another party; Ψ is a unique within $\hat{A}$ string that is used to identify the created session within $\hat{A}$; and $role$ is either initiator $\mathcal{I}$ or responder $\mathcal{R}$. Upon receiving *Create*$(\hat{A}, \hat{B}, \Psi, role)$, $\hat{A}$ verifies that no session was previously created with $(\hat{A}, \hat{B}, \Psi, \overline{role})$ for some $\overline{role}$ not necessarily equal to $role$. For a session $\mathrm{S} = (\hat{A}, \hat{B}, \Psi, role)$, $\hat{A}$ is the *owner* and $\hat{B}$ is the *peer* of S; together $\hat{A}$ and $\hat{B}$ are called *partners* of S. Every session S owned by $\hat{A}$ has an associated *session state* that contains only session-specific information related to S, portions of which are labeled secret. A session produces output of the form $(\hat{A}, \hat{B}, \Psi, \kappa)$, where a null value for κ indicates that an error occurred and the session is *aborted*. A non-null value κ is labeled secret and is called the *session key*. Once a session produces local output its corresponding session state is erased from the memory of the session owner. If an execution of the key establishment protocol has two sessions $\mathrm{S}^1 = (\hat{A}, \hat{B}, \Psi_1, role)$ and $\mathrm{S}^2 = (\hat{B}, \hat{A}, \Psi_2, \overline{role})$ such that $\Psi_1 = \Psi_2$, then S^1 and S^2 are called *matching*. A party $\hat{A}$ could also receive an action request *Expire*(S), where the session S produced a session key κ. Upon receiving *Expire*(S), $\hat{A}$ deletes from its memory the session key κ produced by S and S is labeled *expired*.

In the CK01 model the adversary controls *all* communications. Parties submit outgoing messages and action requests to the adversary, who makes decisions about their delivery. The adversary presents parties with incoming messages and action requests, thereby controlling the creation of sessions. The adversary does not have immediate access to a party's private information, however in order to capture possible leakage of private information the adversary is allowed the following queries:

- *SessionStateReveal*(S) — The adversary obtains the information labeled secret in the session state associated with S. A special note is appended to the session and S produces no further output.
- *SessionKeyReveal*(S) — The adversary obtains the session key for a session S, provided that the session holds a session key.
- *Corrupt*(party) — The adversary takes complete control over a party identified via this query and learns all information that is currently in the party's memory. The party produces no further output. Parties against whom the adversary issued this query are called *corrupt*, otherwise they are called *honest*

or *uncorrupted*.

The adversary's goal is to distinguish a session key from a random key. Let $\mathsf{S} = (\hat{A}, \hat{B}, \Psi, *)$ denote a session owned by $\hat{A}$. If the adversary issues *SessionStateReveal*(S) or *SessionKeyReveal*(S), or *Corrupt*($\hat{A}$) before *Expire*(S), including the case in which $\hat{A}$ is corrupted before S is even created, then S is said to be *locally exposed*. If neither S nor its matching session are locally exposed, then S is said to be *fresh*. At any stage during its execution the adversary is allowed to make one special query *Test*(S), where S is a session that produced a session key, is unexpired and is fresh. In response the adversary is then given with equal probability either the session key held by S or a random key. Provided that S remains fresh throughout the adversary's execution, the adversary is said to win the distinguishing game if the adversary guesses correctly whether the key is random or not. Formally (see also [20, Definition 4]), a protocol is called *CK01-secure* if it satisfies the following conditions:

1. if two uncorrupted parties complete matching sessions, then they both output the same key; and
2. the probability that the adversary's guess is correct is no larger that $\frac{1}{2} + \epsilon(\lambda)$, where ϵ is a function that is negligible in the security parameter λ.

2.2.2 Modeling security goals

CK01-secure protocols achieve many of the desirable security goals that were introduced in §1.2.

Since in the CK01 model the adversary controls all communications the adversary observes all messages exchanged between parties. If an adversary could compute a session key for a session S by only observing messages, then that adversary can win the distinguishing game with probability one by selecting S as the test session and comparing the response to the test query with the computed key. Hence a CK01-secure protocol guarantees security against eavesdroppers.

The CK01 model allows the adversary to learn the session keys held by sessions other than the test session and its matching session. If S^1 and S^2 form the same session key and are non-matching, by comparing the responses to *SessionKeyReveal*(S^2) and to *Test*(S^1) an adversary can win the distinguishing game against S^1. Hence KKS attacks, whereby an adversary induces two sessions $\mathsf{S}^1 = (\hat{A}, \hat{B}, \Psi_1, role)$ and $\mathsf{S}^2 = (\hat{A}, \hat{B}, \Psi_2, role)$ to form the same key, where $\Psi_1 \neq \Psi_2$, are thwarted

by a CK01-secure protocol. Note that in this scenario issuing a *SessionKeyReveal*(s^2) query does not destroy freshness of s^1.

Suppose that an adversary can induce sessions $\mathrm{s}^1 = (\hat{A}, \hat{B}, \Psi, role)$ and $\mathrm{s}^2 = (\hat{B}, \hat{C}, \Psi, role)$ to output the same session key κ, where $\hat{A} \neq \hat{C}$; that is the adversary has successfully launched a UKS attack. These sessions are non matching, therefore by issuing *SessionKeyReveal*(s^2) the adversary can win the distinguishing game against s^1. Hence a CK01-secure protocol is UKS resilient.

Recall the replay attack against the signed Diffie-Hellman protocol. The attack can be modeled in the following way: an adversary creates $\mathrm{s}^1 = (\hat{A}, \hat{B}, \Psi_1, role)$ within $\hat{A}$ and records $\hat{A}$'s response. Afterwards, the adversary issues *SessionStateReveal*(s^1) to obtain the secret information in the session state of s^1. Thereafter, the adversary can create a session $\mathrm{s}^2 = (\hat{B}, \hat{A}, \Psi_2, role)$ and present $\hat{B}$ with the incoming message recorded from s^1. If the adversary can compute the session key output by s^2 by using *SessionStateReveal*(s^1)'s response, then the adversary can win the distinguishing game against s^2. Hence, if a protocol is CK01-secure, then leakage of session-specific ephemeral information does not compromise other sessions.

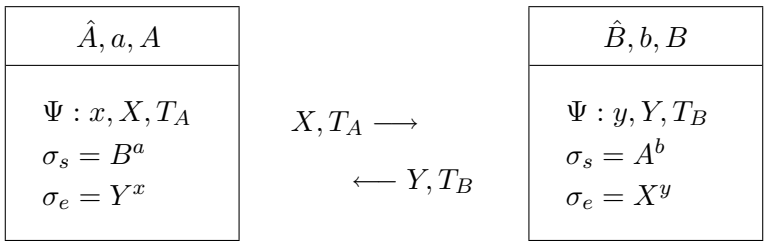
Suppose that an adversary could perform the following sequence of actions. First the adversary creates a session $\mathrm{s} = (\hat{A}, \hat{B}, \Psi, role)$ within $\hat{A}$, and s produces a session key κ. Then the adversary issues *Test*(s), *Expire*(s) and *Corrupt*($\hat{A}$) in that order. If at the end the adversary is able to compute κ, then the adversary can win the distinguishing game against s. Therefore, a CK01-secure protocol provides forward secrecy.

2.3 A CK01-secure protocol

The μ-protocol, informally depicted in Figure 2.1, was first presented in [54], without a formal security argument. It is an example of a Diffie-Hellman type protocol and its formal description, given in Definition 2.3.1, fits into the CK01 model.

2.3.1 Protocol description

Let λ denote the security parameter. In the protocol description, $\mathcal{H} : \{0,1\}^* \rightarrow \{0,1\}^\lambda$ and $\mathcal{H}_2 : \{0,1\}^* \rightarrow \{0,1\}^\lambda$ are random oracles and $\hat{A}$ and $\hat{B}$ are two parties with valid public-private key pairs (A, a) and (B, b), respectively. The μ-protocol is formally given in the following definition:



$$T_A = \mathcal{H}_2(\sigma_s, \Psi, X, \hat{A}, \hat{B}, \mathcal{I}) \qquad T_B = \mathcal{H}_2(\sigma_s, \Psi, Y, \hat{B}, \hat{A}, \mathcal{R})$$

$$\kappa = \mathcal{H}(\sigma_e, X, Y)$$

Figure 2.1: μ-protocol

Definition 2.3.1 (μ-protocol) *The protocol proceeds as follows:*

1. *Upon activation $(\hat{A}, \hat{B}, \Psi, \mathcal{I})$, $\hat{A}$ (the initiator) performs the steps:*
   (a) *Create a session with identifier $(\hat{A}, \hat{B}, \Psi, \mathcal{I})$, provided that no session with identifier $(\hat{A}, \hat{B}, \Psi, *)$ exists.*
   (b) *Select an ephemeral private key $x \in_R \mathbb{Z}_q^*$ and compute the corresponding ephemeral public key $X = g^x$.*
   (c) *Compute $\sigma_s = B^a$ and commitment for X, $T_A = \mathcal{H}_2(\sigma_s, \Psi, X, \hat{A}, \hat{B}, \mathcal{I})$.*
   (d) *Destroy σ_s and send $(\hat{B}, \hat{A}, \Psi, \mathcal{R}, X, T_A)$ to $\hat{B}$.*
2. *Upon activation $(\hat{B}, \hat{A}, \Psi, \mathcal{R}, X, T_A)$, $\hat{B}$ (the responder) performs the steps:*
   (a) *Create a session with identifier $(\hat{B}, \hat{A}, \Psi, \mathcal{R})$, provided that no session with identifier $(\hat{B}, \hat{A}, \Psi, *)$ exists.*
   (b) *Verify that $X \in \mathcal{G}^*$.*
   (c) *Compute $\sigma_s = A^b$ and verify that $T_A = \mathcal{H}_2(\sigma_s, \Psi, X, \hat{A}, \hat{B}, \mathcal{I})$.*
   (d) *Select an ephemeral private key $y \in_R \mathbb{Z}_q^*$ and compute the corresponding ephemeral public key $Y = g^y$.*
   (e) *Compute commitment for Y, $T_B = \mathcal{H}_2(\sigma_s, \Psi, Y, \hat{B}, \hat{A}, \mathcal{R})$.*
   (f) *Compute the session key $\kappa = \mathcal{H}(X^y, X, Y)$.*
   (g) *Destroy σ_s and y, and send $(\hat{A}, \hat{B}, \Psi, \mathcal{I}, Y, T_B)$ to $\hat{A}$.*
   (h) *Complete the session $(\hat{B}, \hat{A}, \Psi, \mathcal{R})$ with output $(\hat{B}, \hat{A}, \Psi, \kappa)$*
3. *Upon activation $(\hat{A}, \hat{B}, \Psi, \mathcal{I}, Y, T_B)$, $\hat{A}$ performs the steps:*

(a) *Verify that* $(\hat{A}, \hat{B}, \Psi, \mathcal{I})$ *exists and* $Y \in \mathcal{G}^*$.

(b) *Compute* $\sigma_s = B^a$ *and verify that* $T_B = \mathcal{H}_2(\sigma_s, \Psi, Y, \hat{B}, \hat{A}, \mathcal{R})$.

(c) *Compute the session key* $\kappa = \mathcal{H}(Y^x, X, Y)$.

(d) *Destroy* σ_s *and* x.

(e) *Complete session* $(\hat{A}, \hat{B}, \Psi, \mathcal{I})$ *with output* $(\hat{A}, \hat{B}, \Psi, \kappa)$.

If any of the verifications fail, the party erases all session-specific information, which includes the identities $\hat{A}$ *and* $\hat{B}$*, and the corresponding ephemeral private key.*

Henceforth, it is assumed that the adversary cannot issue a *SessionStateReveal*, *SessionKeyReveal* or *Corrupt* query while a party is executing one of the three main steps of the protocol. That is, the adversary can only issue one of these queries at the end of Steps 1, 2 or 3.

2.3.2 Security argument

The security of the μ-protocol is established by the following theorem:

Theorem 2.3.1 *If* $\mathcal{H}$ *and* $\mathcal{H}_2$ *are modeled as random oracles, and* $\mathcal{G}$ *is a group where the GDH assumption holds, then the* μ*-protocol is CK01-secure.*

Proof: Let λ denote the security parameter and $\mathcal{M}$ be a polynomially (in λ) bounded adversary. Hence $q = \Theta(2^\lambda)$. The adversary $\mathcal{M}$ is said to be successful (event M) with non-negligible probability if $\mathcal{M}$ wins the distinguishing game with probability $1/2 + p(\lambda)$, where $p(\lambda)$ is a non-negligible term. Assume further that $\mathcal{M}$ succeeds in an environment with $\mathbf{n}(\lambda)$ parties, activates at most $\mathbf{s}(\lambda)$ sessions within each party, makes at most $h_2(\lambda)$ and $h(\lambda)$ queries to the oracles $\mathcal{H}_2$ and $\mathcal{H}$ respectively, and terminates after time at most $\mathcal{T}_{\mathcal{M}}(\lambda)$. Following the standard approach, such a μ-protocol adversary $\mathcal{M}$ is used to construct a GDH solver $\mathcal{S}$, that succeeds with non-negligible probability. Let $\xi : \mathcal{G} \times \mathcal{G} \rightarrow \mathcal{G}$ be a random function known only to $\mathcal{S}$ such that $\xi(X, Y) = \xi(Y, X)$. The algorithm $\mathcal{S}$ will use ξ to simulate $\mathrm{CDH}(X, Y)$ when $\mathcal{S}$ may not know $\log_g(X)$ or $\log_g(Y)$. Let (U, V) be the input to the GDH challenge and consider the following complementary events:

ME: The test session has a matching session.

$\overline{ME}$: No session is matching to the test session.

Simulation in event ME. In this scenario $\mathcal{S}$ establishes $\mathbf{n}(\lambda)$ parties, who are assigned random static key pairs and selects $s_1, s_2 \in_R [1, \ldots, \mathbf{ns}]$. The s_1'th and the s_2'th sessions created will be called s^U and s^V, respectively. The simulation of $\mathcal{M}$'s environment proceeds as follows:

1. *Create*$(\hat{A}, \hat{B}, \Psi, \mathcal{I})$: $\mathcal{S}$ executes Step 1 of the protocol. However, if the session being created is the s_1'th or the s_2'th session, then $\mathcal{S}$ deviates from the protocol by setting the ephemeral public key X to be U or V, respectively; note that $\mathcal{S}$ does not possess the corresponding ephemeral private key in this case.

2. *Create*$(\hat{B}, \hat{A}, \Psi, \mathcal{R}, X, T_A)$: $\mathcal{S}$ executes Step 2 of the protocol. However, if the session being created is the s_1'th or the s_2'th session, then $\mathcal{S}$ deviates from the protocol by setting the ephemeral public key Y to be U or V, respectively and setting $\sigma_e = \xi(X, Y)$; note that $\mathcal{S}$ does not possess the corresponding ephemeral private key in this case.

3. *Message*$(\hat{A}, \hat{B}, \Psi, \mathcal{I}, Y, T_B)$: $\mathcal{S}$ executes Step 3 of the protocol. However, if $(\hat{A}, \hat{B}, \Psi, \mathcal{I}) \in \{\mathrm{s}^U, \mathrm{s}^V\}$, then $\mathcal{S}$ deviates by setting $\sigma_e = \xi(X, Y)$, where X is $\hat{A}$'s ephemeral public key.

4. *SessionStateReveal*(s): If $\mathrm{s} \in \{\mathrm{s}^U, \mathrm{s}^V\}$, then $\mathcal{S}$ aborts, otherwise $\mathcal{S}$ answers the query faithfully.

5. *SessionKeyReveal*(s): If $\mathrm{s} \in \{\mathrm{s}^U, \mathrm{s}^V\}$, then $\mathcal{S}$ aborts with failure, otherwise $\mathcal{S}$ answers the query faithfully.

6. *Corrupt*$(\hat{A})$: If $\hat{A}$ owns either s^U or s^V and that session is not yet expired, then $\mathcal{S}$ aborts with failure, otherwise; $\mathcal{S}$ answers the query faithfully.

7. *Expire*(s): $\mathcal{S}$ answers the query faithfully.

8. $\mathcal{H}_2(*)$: $\mathcal{S}$ simulates a random oracle in the usual way by returning random values for new queries and replaying answers if the queries were previously made.

9. $\mathcal{H}(\sigma_e, X, Y)$:

   (a) If $X \in \{U, V\}$ and $\sigma_e \neq \xi(X, Y)$, then $\mathcal{S}$ obtains $\tau = \mathrm{DDH}(X, Y, \sigma_e)$.

      i. If $\tau = 0$, then $\mathcal{S}$ simulates a random oracle in the usual way.

      ii. If $\tau = 1$ and $Y \in \{U, V\}$ and $Y \neq X$, then $\mathcal{S}$ aborts with success and outputs $\mathrm{CDH}(U, V) = \sigma_e$. Otherwise, if either $Y \notin \{U, V\}$ or $Y = X$, then $\mathcal{S}$ returns $\mathcal{H}(\xi(X, Y), X, Y)$

(b) If $Y \in \{U, V\}$ and $\sigma_e \neq \xi(X, Y)$, then $\mathcal{S}$ obtains $\tau = \mathrm{DDH}(X, Y, \sigma_e)$.

i. If $\tau = 0$, then $\mathcal{S}$ simulates a random oracle in the usual way.

ii. If $\tau = 1$, then $\mathcal{S}$ returns $\mathcal{H}(\xi(X, Y), X, Y)$

(c) $\mathcal{S}$ simulates a random oracle in the usual way.

10. *Test*(s^t): If $\mathrm{s}^t \notin \{\mathrm{s}^U, \mathrm{s}^V\}$ or if s^U and s^V are not matching, then $\mathcal{S}$ aborts with failure; otherwise $\mathcal{S}$ answers the query faithfully.

11. $\mathcal{M}$ outputs a guess: $\mathcal{S}$ aborts with failure.

Analysis of event $M \wedge ME$. Suppose that event $M \wedge ME$ occurs. With probability at least $2/(\mathbf{sn})^2$, $\mathcal{S}$ correctly guesses the test session and its matching, in which case $\mathcal{S}$ does not abort as in Step 10. Without loss of generality, let $\mathrm{s}^t = \mathrm{s}^U = (\hat{A}, \hat{B}, \Psi, \mathcal{I})$. Event M implies that the test session remains fresh throughout the experiment and hence $\mathcal{S}$ does not abort as described in Steps 4, 5 and 6. Let H be the event that $\mathcal{M}$ queries $\mathcal{H}(\mathrm{CDH}(U, V), U, V)$ and let $\overline{H}$ be the complement of H. The only way the adversary $\mathcal{M}$ can obtain from the parties the input or the output of the query $\mathcal{H}(\mathrm{CDH}(U, V), U, V)$ is to issue *SessionStateReveal*(s) or *SessionKeyReveal*(s) for $\mathrm{s} \in \{\mathrm{s}^U, \mathrm{s}^V\}$, or to issue *Corrupt*($\hat{A}$) or *Corrupt*($\hat{B}$) before expiring s^U or s^V. Therefore, if event $ME \wedge \overline{H}$ occurs, then $\mathcal{M}$ is successful with probability at most $\frac{1}{2}$. If event $M \wedge ME \wedge H$ occurs, then $\mathcal{S}$ succeeds as described in Step 9a and does not fail as described in Step 11. Consequently,

$$\begin{aligned} \mathcal{P}(M \wedge ME) &= \mathcal{P}(M \mid ME \wedge \overline{H})\mathcal{P}(ME \wedge \overline{H}) + \mathcal{P}(M \wedge ME \wedge H) \\ &\leq \frac{1}{2}\mathcal{P}(ME \wedge \overline{H}) + \mathcal{P}(M \wedge ME \wedge H). \end{aligned} \tag{2.1}$$

Simulation in event $\overline{ME}$. In this scenario $\mathcal{S}$ establishes $\mathbf{n}(\lambda)$ parties. Two of these parties, denoted by $\hat{U}$ and $\hat{V}$, are selected at random and assigned static public keys U and V respectively. The remaining parties are assigned random static key pairs. Note that $\mathcal{S}$ does not know the static private keys of $\hat{U}$ and $\hat{V}$. The simulation of $\mathcal{M}$'s environment proceeds as follows:

1. *Create*($\hat{A}, \hat{B}, \Psi, \mathcal{I}$): $\mathcal{S}$ executes Step 1 of the protocol. However, if $\hat{A} \in \{\hat{U}, \hat{V}\}$, then $\mathcal{S}$ deviates from the protocol by setting $\sigma_s = \xi(A, B)$.

2. *Create*($\hat{B}, \hat{A}, \Psi, \mathcal{R}, X, T_A$): $\mathcal{S}$ executes Step 2 of the protocol. However, if $\hat{B} \in \{\hat{U}, \hat{V}\}$, then $\mathcal{S}$ deviates from the protocol by setting $\sigma_s = \xi(A, B)$.

3. *Message*$(\hat{A}, \hat{B}, \Psi, \mathcal{I}, Y, T_B)$: $\mathcal{S}$ executes Step 3 of the protocol. However, if $\hat{A} \in \{\hat{U}, \hat{V}\}$, then $\mathcal{S}$ deviates from the protocol by setting $\sigma_s = \xi(A, B)$.

4. *SessionStateReveal*(s): $\mathcal{S}$ answers the query faithfully.

5. *SessionKeyReveal*(s): $\mathcal{S}$ answers the query faithfully.

6. *Corrupt*$(\hat{A})$: If $\hat{A} \in \{\hat{U}, \hat{V}\}$, then $\mathcal{S}$ aborts with failure; otherwise $\mathcal{S}$ answers the query faithfully.

7. *Expire*(s): $\mathcal{S}$ answers the query faithfully.

8. $\mathcal{H}_2(\sigma_s, \Psi, X, \hat{A}, \hat{B}, role)$

   (a) If $\hat{A} \in \{\hat{U}, \hat{V}\}$ and $\sigma_e \neq \xi(A, B)$, then $\mathcal{S}$ obtains $\tau = \mathrm{DDH}(A, B, \sigma_s)$.

      i. If $\tau = 0$, then $\mathcal{S}$ simulates a random oracle in the usual way.

      ii. If $\tau = 1$ and $\hat{A} \in \{\hat{U}, \hat{V}\}$ and $\hat{A} \neq \hat{B}$, then $\mathcal{S}$ aborts $\mathcal{M}$ and outputs $\mathrm{CDH}(U, V) = \sigma_e$. Otherwise, if either $\hat{B} \notin \{\hat{U}, \hat{V}\}$ or $\hat{A} = \hat{B}$, then $\mathcal{S}$ returns $\mathcal{H}_2(\xi(A, B), \Psi, X, \hat{A}, \hat{B}, role)$

   (b) If $\hat{B} \in \{\hat{U}, \hat{V}\}$ and $\sigma_s \neq \xi(A, B)$, then $\mathcal{S}$ obtains $\tau = \mathrm{DDH}(A, B, \sigma_s)$.

      i. If $\tau = 0$, then $\mathcal{S}$ simulates a random oracle in the usual way.

      ii. If $\tau = 1$, then $\mathcal{S}$ returns $\mathcal{H}_2(\xi(A, B), \Psi, X, \hat{A}, \hat{B}, role)$

   (c) $\mathcal{S}$ simulates a random oracle in the usual way.

9. $\mathcal{H}(*)$: $\mathcal{S}$ simulates a random oracle in the usual way.

10. *Test*(s^t): If the communicating partners of s^t are not $\hat{U}$ and $\hat{V}$, then $\mathcal{S}$ aborts with failure; otherwise $\mathcal{S}$ answers the query faithfully.

11. $\mathcal{M}$ outputs a guess: $\mathcal{S}$ aborts with failure.

Analysis of event $M \wedge \overline{ME}$. Suppose that event $M \wedge \overline{ME}$ occurs. With probability at least $2/\mathbf{n}^2$, $\mathcal{M}$ selects a test session with communicating partners $\hat{U}$ and $\hat{V}$, in which case $\mathcal{S}$ does not abort as in Step 10. Without loss of generality let $\mathrm{s}^t = (\hat{U}, \hat{V}, \Psi, \mathcal{I})$ and let X and Y denote s^t's outgoing and incoming ephemeral public keys, respectively. Under event $M \wedge \overline{ME}$ the test session s^t outputs a session key κ_t. Before κ_t is output within $\hat{U}$, $\hat{U}$ receives T_V that passes the verification procedures described by the protocol. Since the string Ψ is unique within $\hat{U}$ and since s^t has no matching session, no honest party could have generated T_V. Hence, except with the negligible probability of guessing $\xi(U, V)$, a successful adversary must query $\mathcal{H}_2(\mathrm{CDH}(U, V), \Psi, Y, \hat{V}, \hat{U}, \mathcal{R})$ (event H_2) in which case $\mathcal{S}$ is successful as described

in Step 8a and does not abort as described in Steps 6 and 11. Therefore, if event $\overline{ME} \wedge \overline{H_2}$ occurs, then $\mathcal{M}$ is successful with at most negligible probability, which can be bounded by $\frac{1}{2}$. Consequently,

$$\begin{aligned}\mathcal{P}(M \wedge \overline{ME}) &= \mathcal{P}(M \mid \overline{ME} \wedge \overline{H_2})\mathcal{P}(ME \wedge \overline{H_2}) + \mathcal{P}(M \wedge \overline{ME} \wedge H_2) \\ &\leq \frac{1}{2}\mathcal{P}(\overline{ME} \wedge \overline{H_2}) + \mathcal{P}(M \wedge \overline{ME} \wedge H_2). \end{aligned} \tag{2.2}$$

Analysis of event M. Let S denote the event that $\mathcal{S}$ outputs a solution to the GDH instance (U, V) and let $p_{ME} = \mathcal{P}(M \wedge ME \wedge H)$ and $p_{\tilde{ME}} = \mathcal{P}(M \wedge \overline{ME} \wedge H_2)$. Suppose that event M occurs with non-negligible probability. From the analysis of event $M \wedge ME$ it follows that S occurs with probability at least $\frac{1}{(\mathbf{sn})^2} p_{ME}$ and from the analysis of event $M \wedge \overline{ME}$ it follows that S occurs with probability at least $\frac{1}{\mathbf{n}^2} p_{\tilde{ME}}$. Since events ME and $\overline{ME}$ are complementary it follows that

$$\mathcal{P}(S) \geq \min\left\{\frac{1}{(\mathbf{sn})^2} p_{ME}, \frac{1}{\mathbf{n}^2} p_{\tilde{ME}}\right\}. \tag{2.3}$$

During the simulation $\mathcal{S}$ performs group exponentiations, accesses the DDH oracle, and simulates random oracles. A group exponentiation takes polynomial time $\mathcal{T}_{\mathcal{G}}(\lambda)$. Assume that a DDH oracle call and response to a random oracle query take polynomial time $\mathcal{T}_{DDH}(\lambda)$ and $\mathcal{T}_{\mathcal{H}}(\lambda)$, respectively. The running time $\mathcal{T}_{\mathcal{S}}$ of $\mathcal{S}$ is therefore bounded by

$$\mathcal{T}_{\mathcal{S}} \leq (2\mathcal{T}_{\mathcal{G}} + \mathcal{T}_{\mathrm{DDH}} + \mathcal{T}_{\mathcal{H}})\, \mathcal{T}_{\mathcal{M}}. \tag{2.4}$$

Equations 2.1 and 2.2 imply

$$\begin{aligned}\mathcal{P}(M) &= \mathcal{P}(M \wedge ME) + \mathcal{P}(M \wedge \overline{ME}) \\ &\leq p_{ME} + p_{\tilde{ME}} + \frac{1}{2}\left(\mathcal{P}(ME \wedge \overline{H}) + \mathcal{P}(\overline{ME} \wedge \overline{H_2})\right).\end{aligned}$$

From $\mathcal{P}(ME \wedge \overline{H}) \leq \mathcal{P}(ME)$ and $\mathcal{P}(\overline{ME} \wedge \overline{H_2}) \leq \mathcal{P}(\overline{ME})$ it follows that

$$\mathcal{P}(M) \leq p_{ME} + p_{\tilde{ME}} + \frac{1}{2}\left(\mathcal{P}(ME) + \mathcal{P}(\overline{ME})\right) = p_{ME} + p_{\tilde{ME}} + \frac{1}{2},$$

which implies $p \leq p_{ME} + p_{\tilde{ME}}$. Therefore, if p is non-negligible, then either p_{ME} or $p_{\tilde{ME}}$ is non-negligible, and hence by Equations 2.3 and 2.4 it follows that $\mathcal{S}$ is a polynomially bounded GDH solver contradicting the GDH assumption in $\mathcal{G}$. Therefore, the μ-protocol is a CK01-secure protocol, concluding the proof of Theorem 2.3.1. □

The following numbers illustrate the non-tightness of the above argument. Suppose that the desired security level is 80 bits. Suppose also $\mathbf{n} = 2^{15}$ and $\mathbf{s} = 2^{20}$. Assume that the best algorithm to solve GDH problem in $\mathcal{G}$ takes approximately $\sqrt{q}$ steps and that $p_{ME} \approx p_{\tilde{ME}}$. To achieve $\lambda = 80$, then $q \approx 2^{300}$.

2.4 Critique of the CK01 model

This section outlines shortcomings of the CK01 model.

Modeling KCI attacks requires that the adversary be given the ability to create a new session within $\hat{A}$ *after* obtaining $\hat{A}$'s static private key a. In the CK01 model the adversary obtains a via *Corrupt*($\hat{A}$), at which stage $\hat{A}$ becomes corrupted and does not produce further output. In particular, the adversary cannot create a new session within $\hat{A}$. Thus the CK01 model does not cover KCI attacks.

In some scenarios, a party $\hat{A}$ may desire to establish a shared secret key with itself (reflections). In the CK01 language $\hat{A}$ owns $\mathrm{S}^1 = (\hat{A}, \hat{A}, \Psi, \mathcal{I})$ and $\mathrm{S}^2 = (\hat{A}, \hat{A}, \Psi, \mathcal{R})$. However, if S^1 is created, then the CK01 model prohibits the creation of S^2. Thus the CK01 model does not capture reflection attacks.

The CK01 model does not allow *malicious insiders*. In particular the malicious insider UKS attack described in §1.2 against the basic MTI protocol is not covered. Indeed the attack in §1.2 relies on the fact that the adversary is able to establish a new static public key for an entity, which is not allowed by the CK01 model.

Nowadays a typical Diffie-Hellman key agreement protocol combines the session partners' ephemeral and static key pairs to derive a shared secret key. Let $\hat{A}$ and $\hat{B}$ be partners of a session S. Let x and a denote the ephemeral and static private keys of $\hat{A}$, respectively, and let y and b denote the ephemeral and static private keys of $\hat{B}$. Via *SessionStateReveal* the CK01 model captures the security implications of leaking ephemeral private keys of sessions different from S. However, a session S of the basicUM protocol appears to be secure even if the adversary obtains x and y used in S itself. Ideally, any subset of (x, a, y, b) that does not contain either (x, a) or (y, b) should not be sufficient to compute the corresponding session key. In the CK01 model the adversary is *only* allowed to obtain the pair (a, b) *after* S and its matching session are expired.

Choo, Boyd and Hitchcock [23] compared the most commonly used security models for key agreement [9, 8, 20]. Their conclusion was that none of the models as defined provides a significant advantage over the rest of the models. Furthermore, these models fail to capture some desirable properties of key agreement. Most significantly, the adversary is not allowed to obtain certain secret information about the session that is being attacked. Krawczyk [41] addressed many of these concerns by providing a stronger version of the Canetti-Krawczyk model [20] that captures additional security properties. These desirable properties include resistance to KCI attacks, resilience to LEP attacks, and malicious insiders. More recently LaMacchia, Lauter and Mityagin [45] provided a single definition that *si-*

multaneously captures all these security properties. Their security model will henceforth be called the extended Canetti-Krawczyk (eCK) model.

2.5 Extended Canetti-Krawzcyk security model

In the eCK model there are n parties each modeled by a probabilistic Turing machine. Each party has a static public-private key pair together with a certificate that binds the public key to that party. The certifying authority (CA) does not require parties to prove possession of their static private keys, but the CA verifies that the static public key of a party belongs to $\mathcal{G}^*$. For simplicity, the model is described only for two-pass Diffie-Hellman protocols that exchange ephemeral and static public keys. More precisely, two parties $\hat{A}$, $\hat{B}$ exchange static public keys $A, B \in \mathcal{G}^*$ and ephemeral public keys $X, Y \in \mathcal{G}^*$; the session key is obtained by combining A, B, X, Y and possibly the identities $\hat{A}$, $\hat{B}$.

A party $\hat{A}$ can be activated to execute an instance of the protocol called a *session*. Activation is made via an incoming message that has one of the following forms: (i) $(\hat{A}, \hat{B})$ or (ii) $(\hat{A}, \hat{B}, Y)$. If $\hat{A}$ was activated with $(\hat{A}, \hat{B})$, then $\hat{A}$ is the session *initiator*, otherwise the session *responder*. If $\hat{A}$ is the responder of a session, then $\hat{A}$ prepares an ephemeral public key X and creates a separate session *state* where all session-specific ephemeral information is stored. The session is identified via a session *identifier* $(\hat{A}, \hat{B}, X, Y)$. If $\hat{A}$ is the initiator of a session, then $\hat{A}$ prepares an ephemeral public key X and creates a session state as in the responder case. At the onset of the protocol the initiator does not know the incoming ephemeral public key. Since ephemeral keys are selected at random on a per-session basis, the probability that an ephemeral public key X is chosen twice by $\hat{A}$ is negligible; hence the session can be uniquely identified with $(\hat{A}, \hat{B}, X, \times)$, and consequently this string can be used as the (temporary and incomplete) session identifier. When $\hat{A}$ receives the corresponding ephemeral public key Y, the session identifier is updated to $(\hat{A}, \hat{B}, X, Y)$. A session $(\hat{B}, \hat{A}, Y, X)$ (if it exists) is said to be *matching* to the session $(\hat{A}, \hat{B}, X, \times)$. On the other hand, the session matching to $(\hat{A}, \hat{B}, X, Y)$ can be any session identified by $(\hat{B}, \hat{A}, Y, \times)$ or $(\hat{B}, \hat{A}, Y, X)$. Since it is not possible (except with negligible probability) to simultaneously have two different sessions with identifiers $(\hat{B}, \hat{A}, Y, \times)$ and $(\hat{B}, \hat{A}, Y, X)$, a session $(\hat{A}, \hat{B}, X, Y)$ can have at most one matching session. For a session $(\hat{A}, \hat{B}, *, *)$, $\hat{A}$ is the session *owner* and $\hat{B}$ is the session *peer*.

The adversary $\mathcal{M}$ is modeled as a probabilistic Turing machine and controls *all* communications. Parties submit outgoing messages to the adversary, who makes

decisions about their delivery. The adversary presents parties with incoming messages via *Send*(message), thereby controlling the activation of sessions. The adversary does not have immediate access to a party's private information, however in order to capture possible leakage of private information the adversary is allowed the following queries:

- *EphemeralKeyReveal*(s) — The adversary obtains the ephemeral private key held by the session s.
- *SessionKeyReveal*(s) — The adversary obtains the session key for a session s, provided that the session holds a session key.
- *StaticKeyReveal*(party) — The adversary learns the static private key of the party.
- *Establish*(party) — This query allows the adversary to register a static public key on behalf of a party. In this way the adversary totally controls that party. Parties against whom the adversary did not issue this query are called *honest*.

The aim of the adversary $\mathcal{M}$ is to distinguish a session key from a random key. Formally, the adversary is allowed to make one special query *Test*(s). The adversary is then given with equal probability either the session key held by s or a random key. The adversary wins the game if he guesses correctly whether the key is random or not. The eCK security notion requires the following.

Definition 2.5.1 (fresh session) *Let* s *be the session identifier of a completed session, owned by an honest party* $\hat{A}$ *with peer* $\hat{B}$*, who is also honest. Let* s^* *be the session identifier of the matching session of* s*, if it exists. Define* s *to be* fresh *if none of the following conditions hold:*

(i) $\mathcal{M}$ *issues a* SessionKeyReveal(s) *query or a* SessionKeyReveal(s^*) *query (if* s^* *exists);*

(ii) s^* *exists and* $\mathcal{M}$ *makes either of the following queries:*

- *both* StaticKeyReveal($\hat{A}$) *and* EphemeralKeyReveal(s)*, or*
- *both* StaticKeyReveal($\hat{B}$) *and* EphemeralKeyReveal(s^*)*;*

(iii) s^* *does not exist and* $\mathcal{M}$ *makes either of the following queries:*

- *both* StaticKeyReveal($\hat{A}$) *and* EphemeralKeyReveal(s)*, or*

– *StaticKeyReveal*($\hat{B}$).

The eCK security notion is given in the following definition:

Definition 2.5.2 (eCK security) *A key agreement protocol is* eCK-secure *if the following conditions hold:*

1. *If two honest parties complete matching sessions, then, except with negligible probability, they both compute the same session key (or both output indication of protocol failure).*
2. *No polynomially bounded adversary $\mathcal{M}$ can distinguish the session key of a fresh session from a randomly chosen session key, with probability greater than 1/2 plus a negligible fraction.*

The adversary $\mathcal{M}$ is allowed to continue interacting with the parties even after issuing the *Test* query. However, the test session must remain fresh throughout the experiment. This security definition is very strong in the sense that it simultaneously captures most of the desirable security properties for authenticated key agreement that have been identified in the literature including resistance to key-compromise impersonation attacks, weak perfect forward secrecy, and resilience to the leakage of ephemeral private keys. Unlike in the CK01 model [20], the adversary in the eCK model is not equipped with a *SessionStateReveal* query which enables it to learn the entire session state of a particular session. This does not represent a deficiency in the eCK model since protocols such as HMQV [41] proven secure in the CK01 model typically specify that the ephemeral private key is the only private information stored in the session state in which case the *EphemeralKeyReveal* query is functionally equivalent to the *SessionStateReveal* query. In general, by specifying that the session specific private information (the session state) is part of the ephemeral private key, the *SessionStateReveal* and *EphemeralKeyReveal* queries can be made functionally equivalent.

Chapter 3

The CMQV protocol

3.1 Motivation

The previous chapter recounted the extended Canetti-Krawzcyk security model. It is natural to look for efficient protocols that are secure in eCK. Even though two-pass protocols are usually in the core of a given family of protocols, one-pass protocols are important in scenarios where one of the parties is not on-line. Typically, security arguments are presented for two-pass protocols. Extending these arguments to variants with three or more rounds may be an easy task, but considering security of the one-pass variant is not as straightforward. In particular there are security properties that no one-pass protocol can provide. This chapter introduces a "combined" MQV protocol — the CMQV key establishment scheme and provides reductionist security arguments for the one-pass and two-pass CMQV protocols in the eCK model.

3.2 The MQV protocol

The MQV [47] protocol has been standardized in the NIST special publication 800-56A [71] as one of the next generation key establishment schemes to be adopted by the US government. For simplicity, the SP800-56A MQV variant will be referred to as *the* MQV protocol. In the two-pass MQV protocol two parties $\hat{A}$ and $\hat{B}$, that wish to agree on a session key, exchange static and ephemeral public keys. Party $\hat{A}$ computes $\sigma = (YB^{\mathrm{E}})^{x+\mathrm{D}a}$ and party $\hat{B}$ computes $\sigma = (XA^{\mathrm{D}})^{y+\mathrm{E}b}$. Here the *public exponents* D and E are derived from X and Y, respectively and have bitlengths that are half the bitlength of q. Thereafter, $\hat{A}$ and $\hat{B}$ compute the session key $\kappa = \mathcal{H}(\sigma, \hat{A}, \hat{B})$.

The MQV protocol requires 2.5 exponentiations per party, which is almost the eDH efficiency of 2 exponentiations per party. Not only is MQV efficient, but it also provides most of the desirable key establishment attributes. For example, MQV has a natural one-pass variant, it is UKS resilient, and the leakage of any non-trivial pair (a trivial pair is a pair of ephemeral and static private key of the same party) from (x, a, y, b) is not sufficient to compute κ. Even though the CDH assumption is necessary for MQV security, it is not known whether it is also sufficient. Kunz-Jacques and Pointcheval [44] showed that MQV security is equivalent to a modification of the CDH assumption called the f-RCDH assumption, but f-RCDH may potentially be *weaker* than the CDH assumption, in the sense that an efficient reduction is known from f-RCDH to CDH, but not from CDH to f-RCDH. Furthermore, the reductionist security argument by Kunz-Jacques and Pointcheval is in a restricted model where, for example, there are only two parties. It remains an open problem to provide a formal security argument for MQV using the CDH assumption in a full security model such as the eCK model.

3.3 Related work

Researchers from IBM and Microsoft have recently proposed two-pass Diffie-Hellman authenticated key agreement protocols called KEA+ [46], NAXOS [45] and HMQV [41]. In these protocols the two communicating parties $\hat{A}$ and $\hat{B}$ exchange static and ephemeral public key pairs, and thereafter combine them to obtain a session key. The papers [41, 46, 45] highlight certain security issues with previous related key agreement protocols and propose solutions to address those issues.

The KEA+ protocol is derived from the KEA [58] protocol developed by the National Security Agency (see also [13, Protocol 4]). In KEA+, $\hat{A}$ and $\hat{B}$ compute $\sigma_1 = g^{ay}$ and $\sigma_2 = g^{xb}$, and thereafter compute a session key $\kappa = \mathcal{H}(\sigma_1, \sigma_2, \hat{A}, \hat{B})$. Lauter and Mityagin [46] provide a security argument for KEA+ in a model that is weaker than eCK; for example the adversary is not allowed to obtain the static private keys of both communicating parties. However, the KEA+ security argument requires only the GDH and RO assumptions, and guarantees KCI resilience.

In a typical Diffie-Hellman ephemeral key exchange a party selects an ephemeral private key that equals the discrete logarithm of the corresponding ephemeral public key. In the NAXOS protocol, $\hat{A}$ selects ephemeral private key $\tilde{x}$ and computes the corresponding ephemeral public key $X = g^{\mathcal{H}_1(\tilde{x}, a)}$. This method of computing ephemeral private-public key pairs will henceforth be called the *NAXOS*

trick. Modulo the ephemeral public key computation, the NAXOS protocol can be viewed as a KEA+ extension. In addition to σ_1 and σ_2, $\hat{A}$ and $\hat{B}$ also compute $\sigma_e = g^{\mathcal{H}_1(\tilde{x},a)\mathcal{H}_1(\tilde{y},b)}$; the session key is $\kappa = \mathcal{H}(\sigma_1, \sigma_2, \sigma_e, \hat{A}, \hat{B})$. The NAXOS trick allows for a relatively simple security argument for the NAXOS protocol in the eCK model. However, NAXOS requires 4 exponentiations per party, and hence it is less efficient than the MQV and KEA+ protocols.

The HMQV protocol, see §6.3.1 for a detailed description, is a *hashed* MQV variant. In HMQV the public exponents are $\text{E} = \mathcal{H}_2(Y, \hat{A})$ and $\text{D} = \mathcal{H}_2(X, \hat{B})$, and like the MQV protocol the public exponents have bitlength that are half the bitlength of q. Unlike MQV, HMQV does not include the identities of the communicating partners in the key derivation function. But most importantly, the HMQV protocol does not mandate public key validation (i.e., verification that public keys belong to $\mathcal{G}^*$); thereby, in groups where validation requires a group exponentiation HMQV is more efficient than MQV. Furthermore, the HMQV protocol has a formal security proof [41] in the CK01 model. Krawzcyk [41] argued that no two-pass protocol can provide FS, but only a weaker version of FS called *weak forward secrecy* (wFS) [8]. A protocol provides wFS if an adversary who only observes an execution of the protocol cannot compute the session key even if the adversary learns the static private keys of the communicating partners after the session completes. In addition to CK01 security, the HMQV security argument in [41] guarantees KCI and LEP resilience, and also wFS, but it is extremely long and complicated, and some significant (but fixable) flaws [52, 53] have been discovered.

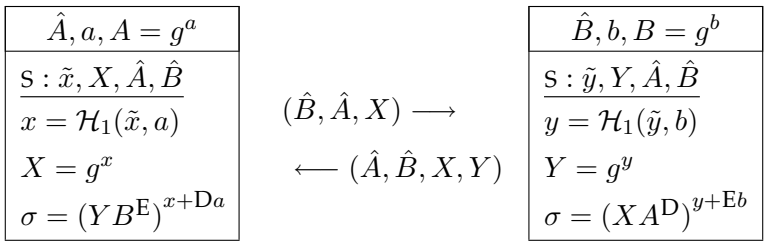
Protocol	Efficiency	Security	Assumptions
MQV	2.5	unproven	?
KEA+	3	CK01, KCI	GDH, RO
NAXOS	4	eCK	GDH, RO
HMQV	2.5	CK01, wFS, KCI, LEP	KEA1, GDH, RO

Table 3.1: Protocol comparison

Currently only NAXOS is shown secure in the eCK model. NAXOS is less efficient in that it requires 4 exponentiations per party compared to 2.5 exponentiations for MQV and HMQV. In addition there is no natural modification of NAXOS to a one-pass protocol. Table 3.1 compares MQV, HMQV, KEA+ and NAXOS in terms of efficiency (number of exponentiations per party), security and underlying assumptions. KEA1 stands for the *knowledge of exponent* [7] assumption. It is natural to ask if there is any protocol that combines the ideas from KEA+, NAXOS, MQV and HMQV, and achieves the best of all worlds.

3.4 Two-pass CMQV

The two-pass CMQV protocol, informally depicted in Figure 3.1, achieves the following objectives: (i) intuitive design principles; (ii) the efficiency of MQV and HMQV; (iii) a relatively straightforward security proof with minimal assumptions in the eCK model; and (iv) a natural one-pass variant. The security proof was inspired by the HMQV argument [41], however the NAXOS trick is essential to show eCK security. Moreover, unlike the HMQV proof, the CMQV security argument does not need the KEA1 assumption in order to demonstrate resilience to the leakage of ephemeral private keys. On the negative side, the security reduction of CMQV is *not* tight. As in the case of HMQV, the reduction uses the Forking Lemma of Pointcheval and Stern [61, 62], which results in a highly non-tight reduction.



$$\mathrm{E} = \mathcal{H}_2(Y, \hat{A}, \hat{B}) \qquad \mathrm{D} = \mathcal{H}_2(X, \hat{A}, \hat{B})$$
$$\kappa = \mathcal{H}(\sigma, X, Y, \hat{A}, \hat{B})$$

Figure 3.1: Two-pass CMQV

3.4.1 Protocol description

In the protocol description, λ is the security parameter; $\mathcal{H}_1 : \{0,1\}^\lambda \times \mathbb{Z}_q^* \to \mathbb{Z}_q^*$, $\mathcal{H}_2 : \{0,1\}^* \to \mathbb{Z}_q$ and $\mathcal{H} : \{0,1\}^* \to \{0,1\}^\lambda$ are random oracles; and $\hat{A}$ and $\hat{B}$ are two parties with valid public-private key pairs (A, a) and (B, b), respectively. The two-pass CMQV protocol is formally given in the following definition:

Definition 3.4.1 (two-pass CMQV protocol) *The protocol proceeds as follows:*

1. *Upon activation $(\hat{A}, \hat{B})$, party $\hat{A}$ (the initiator) performs the steps:*
   (a) *Select an ephemeral private key $\tilde{x} \in_R \{0,1\}^\lambda$.*
   (b) *Compute the ephemeral public key $X = g^{\mathcal{H}_1(\tilde{x}, a)}$.*
   (c) *Initiate session $\mathrm{S} = (\hat{A}, \hat{B}, X, *)$ and send $(\hat{B}, \hat{A}, X)$ to $\hat{B}$.*

2. *Upon activation* $(\hat{B}, \hat{A}, X)$*, party* $\hat{B}$ *(the responder) performs the steps:*

   (a) *Verify that* $X \in \mathcal{G}^*$.

   (b) *Select an ephemeral private key* $\tilde{y} \in_R \{0,1\}^\lambda$.

   (c) *Compute the ephemeral public key* $Y = g^{\mathcal{H}_1(\tilde{y},b)}$.

   (d) *Compute* $\mathrm{E} = \mathcal{H}_2(Y, \hat{A}, \hat{B})$ *and* $\mathrm{D} = \mathcal{H}_2(X, \hat{A}, \hat{B})$.

   (e) *Compute* $\sigma = (XA^{\mathrm{D}})^{\mathcal{H}_1(\tilde{y},b)+\mathrm{E}b}$ *and* $\kappa = \mathcal{H}(\sigma, X, Y, \hat{A}, \hat{B})$.

   (f) *Destroy* $\tilde{y}$ *and* σ.

   (g) *Complete session* $\mathrm{S} = (\hat{B}, \hat{A}, X, Y)$ *with session key* κ *and send* $(\hat{A}, \hat{B}, X, Y)$ *to* $\hat{A}$.

3. *Upon activation* $(\hat{A}, \hat{B}, X, Y)$*, party* $\hat{A}$ *performs the steps:*

   (a) *Verify that a session with identifier* $(\hat{A}, \hat{B}, X, \times)$ *exists.*

   (b) *Verify that* $Y \in \mathcal{G}^*$.

   (c) *Compute* $\mathrm{E} = \mathcal{H}_2(Y, \hat{A}, \hat{B})$ *and* $\mathrm{D} = \mathcal{H}_2(X, \hat{A}, \hat{B})$.

   (d) *Compute* $\sigma = (YB^{\mathrm{E}})^{\mathcal{H}_1(\tilde{x},a)+\mathrm{D}a}$ *and* $\kappa = \mathcal{H}(\sigma, X, Y, \hat{A}, \hat{B})$.

   (e) *Destroy* $\tilde{x}$ *and* σ.

   (f) *Complete session* $\mathrm{S} = (\hat{A}, \hat{B}, X, Y)$ *with session key* κ.

If any verification fails the party erases all session specific information, which includes the ephemeral private key, from its memory and aborts the session.

It is straightforward to verify that both parties compute the same shared secret σ, and therefore also the same session key.

3.4.2 Design rationale

This section explains the underlying principles behind the CMQV design.

Public-key validation. Public-key validation (i.e. checking that static and ephemeral public keys belong to $\mathcal{G}^*$) prevents potential invalid-curve [4] and small subgroup attacks [49] (see also [53]). In other words, with validation a party obtains some assurance that computations involving its static private key do not reveal any information about the static private key itself, as long as the underlying group is cryptographically strong.

Hashing ephemeral and static private keys. In Definition 3.4.1 the value $x = \mathcal{H}_1(\tilde{x}, a)$ is *never* stored. Whenever $\mathcal{H}_1(\tilde{x}, a)$ is needed, it is computed. This implies that the session state does not store x. The idea is that without knowing *both* the ephemeral private key $\tilde{x}$ *and* the static private key a, no entity is able to compute the discrete logarithm x of an ephemeral public key X. This elegant idea, first described in [45], allows the protocol to attain resistance to ephemeral private key leakage without resorting to new assumptions like KEA1 (as needed for HMQV [41]).

Rationale for public exponents. Given a computational Diffie-Hellman challenge with inputs $U, V \in_R \mathcal{G}$, knowledge of either of the discrete logarithms of U or V is enough to solve the CDH instance. If an adversary $\mathcal{M}$, given a static public key B, is able to find a group element Y such that $\mathcal{M}$ knows the discrete logarithm of $T = YB^{\mathrm{E}}$, then that $\mathcal{M}$ can impersonate $\hat{B}$ to other parties by computing the shared secret $\sigma = (XA^{\mathrm{D}})^t$ where $t = \log_g T$, thereby impersonating $\hat{B}$ to $\hat{A}$. Defining E to depend on Y ensures that the adversary is not able to compute the discrete logarithm of YB^{E}. Moreover, including the identity of the intended peer in the derivation of E prevents the adversary from potentially benefiting from the replay of Y to two distinct parties $\hat{A}$ and $\hat{C}$. Arguably, $\hat{B}$'s identity in the derivation of E is not needed since σ in any case depends on $\hat{B}$'s static public key B. However, since the CA does not require parties to prove possession of their static private keys, $\mathcal{M}$ may establish a new party with static public key B. Hence $\hat{B}$ is included in E's derivation.

A very similar definition of E was used in HMQV [41]. For both HMQV and CMQV, the public exponents definition is crucial for the security proof, but in both cases the reduction is non-tight. It is worth investigating if the requirements on E and D can be modified to attain a tight security reduction.

Session key derivation. The session key is $\kappa = \mathcal{H}(\sigma, X, Y, \hat{A}, \hat{B})$. The secrecy of σ guarantees that only the intended parties can possibly compute κ. Including identities in the key derivation is a generic way to prevent unknown-key share attacks (see [14]). Furthermore, inclusion of X and Y in the key derivation allows for a simple argument that non-matching sessions have different session keys.

3.4.3 Efficiency comparison

The efficiency comparison in Table 3.1 is simplified; in particular, it does not take into account validation and various speedups that may be applicable. Consider the

following groups of practical interest: (i) DSA-type groups (order-q subgroups of the multiplicative group of prime fields $\mathbb{F}_p$); and (ii) elliptic curves of prime order q or nearly prime order hq. Validation for DSA-type groups requires a full exponentiation; by contrast validating points on elliptic curves of prime order is essentially free. For nearly prime order curves, rather than verifying that the order of a public key is q, parties could use the corresponding public keys raised by the cofactor h. If the two public keys Y and B are validated, then computing $(YB^{\mathrm{E}})^{\mathcal{H}_1(\tilde{x},a)+\mathrm{D}a}$ is equivalent to computing $Y^{s_1}B^{s_2}$, where $s_1 = \mathcal{H}_1(\tilde{x},a) + \mathrm{D}a \bmod q$ and $s_2 = \mathrm{E}(\mathcal{H}_1(\tilde{x},a) + \mathrm{D}a) \bmod q$. Therefore, CMQV computations can be speedup using Shamir's trick [55, Algorithm 14.88], reducing the cost by 0.75 exponentiations on average.

	DSA groups	Elliptic curves of prime order	Elliptic curves of nearly prime order
CMQV	3.25 (4)	2.25 (3)	2.25 (3)
MQV	3.25 (3.5)	2.25 (2.5)	2.25 (2.5)
HMQV-P1363	2.5 / 3.5	2.25	2.25

Table 3.2: Efficiency comparison in terms of group exponentiations

Table 3.2 compares CMQV with HMQV as described in [42], accounting for the validation and Shamir's speedup. The numbers in parentheses for MQV and CMQV represent the naive count of group exponentiations without accounting for possible improvements in the computations. The numbers for HMQV correspond to the two versions of HMQV as described in [42]. For HMQV, the difference is significant only in DSA-type groups as the more efficient version avoids full validation. However, the security proof in the case where validation is not required assumes that no ephemeral private keys are leaked to the adversary.

3.4.4 Security argument

This section presents a formal security argument for two-pass CMQV. For simplicity, the reduction assumes that the protocol does not allow reflections. This condition can be relaxed by using the GSP assumption instead of the GDH assumption

Theorem 3.4.1 *If $\mathcal{H}_1$,$\mathcal{H}_2$ and $\mathcal{H}$ are random oracles, and $\mathcal{G}$ is a group where the GDH assumption holds, then two-pass CMQV is eCK secure.*

Proof: Verifying condition 1 of Definition 2.5.2 is straightforward; it remains to verify condition 2.

Let λ denote the security parameter, whence $q = |\mathcal{G}| = \Theta(2^{\lambda})$. Let $\mathcal{M}$ be a polynomially (in λ) bounded CMQV adversary. The adversary $\mathcal{M}$ is said to be successful (event M) with non-negligible probability if $\mathcal{M}$ wins the distinguishing game described in §2.5 with probability $\frac{1}{2} + p(\lambda)$, where $p(\lambda)$ is non-negligible. Assume that $\mathcal{M}$ operates in an environment that involves at most $\mathbf{n}(\lambda)$ parties, $\mathcal{M}$ activates at most $\mathbf{s}(\lambda)$ sessions within a party, and makes at most $h_1(\lambda), h_2(\lambda)$ and $h(\lambda)$ queries to oracles $\mathcal{H}_1$,$\mathcal{H}_2$ and $\mathcal{H}$, respectively; and terminates after time at most $\mathcal{T}_{\mathcal{M}}$. Let the test session be $\mathrm{s}^t = (\hat{A}, \hat{B}, X, Y)$ and let H denote the event that $\mathcal{M}$ queries $\mathcal{H}$ with $(\sigma, X, Y, \hat{A}, \hat{B})$, where $\sigma = \mathrm{CDH}(XA^{\mathrm{D}}, YB^{\mathrm{E}})$. Let $\overline{H}$ be the complement of H and s^* be any completed session owned by an honest party, such that s^t and s^* are non-matching. Since s^t and s^* are non-matching, the input to the key derivation function $\mathcal{H}$ are different for s^t and s^*. And since $\mathcal{H}$ is a random oracle it follows that $\mathcal{M}$ cannot obtain any information about the test session key from the session key of non-matching sessions. Hence $\mathcal{P}(M \wedge \overline{H}) \leq \frac{1}{2}$ and

$$\mathcal{P}(M) = \mathcal{P}(M \wedge \overline{H}) + \mathcal{P}(M \wedge H) \leq \frac{1}{2} + \mathcal{P}(M \wedge H),$$

whence $\mathcal{P}(M \wedge H) \geq p$; henceforth the event $M \wedge H$ is denoted by M^*.

Following the standard approach such an adversary $\mathcal{M}$ is used to construct a GDH solver $\mathcal{S}$ that succeeds with non-negligible probability. Let $\xi : \mathcal{G} \times \mathcal{G} \rightarrow \mathcal{G}$ be a random function known only to $\mathcal{S}$, such that $\xi(X, Y) = \xi(Y, X)$. The algorithm $\mathcal{S}$ will use ξ to simulate $\mathrm{CDH}(X, Y)$ when $\mathcal{S}$ may not know $\log_g(X)$ or $\log_g(Y)$. Let the input to the GDH challenge be (U, V) and consider the following complementary events:

DL. There exists an honest party $\hat{B}$ such that $\mathcal{M}$, during its execution, queries $\mathcal{H}_1$ with $(*, b)$, before issuing a *StaticKeyReveal*$(\hat{B})$ query. (Note that $\mathcal{M}$ does not necessarily make a *StaticKeyReveal*$(\hat{B})$ query.)

$\overline{DL}$. During its execution, for every honest party $\hat{B}$ for which $\mathcal{M}$ queries $\mathcal{H}_1$ with $(*, b)$, $\mathcal{M}$ issued *StaticKeyReveal*$(\hat{B})$ before the first $(*, b)$ query to $\mathcal{H}_1$.

If $\mathcal{M}$ succeeds with non-negligible probability, and hence $\mathcal{P}(M^*) \geq p$, it must be the case that either event $DL \wedge M^*$ or event $\overline{DL} \wedge M^*$ occurs with non-negligible probability. These events are considered separately.

Simulation in event DL. Suppose that event $DL \wedge M^*$ occurs with non-negligible probability. In this case $\mathcal{S}$ prepares $\mathbf{n}$ parties. One party, called $\hat{V}$, is selected at random and assigned static public key V; $\mathcal{S}$ represents $\hat{V}$'s static private key by $\nu \in_R \mathbb{Z}_q$. The remaining $\mathbf{n}-1$ parties are assigned random static key pairs. The adversary $\mathcal{M}$ is initiated on this set of parties and the simulation of $\mathcal{M}$'s environment proceeds as follows:

1. *Send*$(\hat{A}, \hat{B})$: $\mathcal{S}$ executes Step 1 of the protocol.
2. *Send*$(\hat{B}, \hat{A}, X)$: $\mathcal{S}$ executes Step 2 of the protocol. However, if $\hat{B} = \hat{V}$, then $\mathcal{S}$ sets $\sigma = \xi(XA^{\mathrm{D}}, YB^{\mathrm{E}})$.
3. *Send*$(\hat{A}, \hat{B}, X, Y)$: $\mathcal{S}$ executes Step 3 of the protocol. However, if $\hat{A} = \hat{V}$, then $\mathcal{S}$ sets $\sigma = \xi(XA^{\mathrm{D}}, YB^{\mathrm{E}})$.
4. *EphemeralKeyReveal*(s): $\mathcal{S}$ responds to the query faithfully.
5. *SessionKeyReveal*(s): $\mathcal{S}$ responds to the query faithfully.
6. *StaticKeyReveal*$(\hat{A})$: $\mathcal{S}$ responds to the query faithfully, unless $\hat{A} = \hat{V}$ in which case $\mathcal{S}$ aborts with failure.
7. *Establish*$(\hat{M})$: $\mathcal{S}$ responds to the query faithfully.
8. $\mathcal{H}_1(s, c)$: $\mathcal{S}$ checks if $g^c = V$; if the equation holds, then $\mathcal{S}$ stops $\mathcal{M}$ and computes $\mathrm{CDH}(U, V) = U^c$. In all other cases $\mathcal{S}$ simulates a random oracle in the usual way.
9. $\mathcal{H}_2(*)$: $\mathcal{S}$ simulates a random oracle in the usual way.
10. $\mathcal{H}(\sigma, X, Y, \hat{A}, \hat{B})$:
    (a) If $\hat{V} \in \{\hat{A}, \hat{B}\}$ and $\sigma \neq \xi(XA^{\mathrm{D}}, YB^{\mathrm{E}})$, then $\mathcal{S}$ obtains $\tau = \mathrm{DDH}(XA^{\mathrm{D}}, YB^{\mathrm{E}}, \sigma)$.
        i. If $\tau = 1$, then $\mathcal{S}$ returns $\mathcal{H}\left(\xi(XA^{\mathrm{D}}, YB^{\mathrm{E}}), X, Y, \hat{A}, \hat{B}\right)$.
        ii. If $\tau = 0$, then $\mathcal{S}$ simulates a random oracle in the usual way.
    (b) $\mathcal{S}$ simulates a random oracle in the usual way.
11. *Test*(s): $\mathcal{S}$ responds to the query faithfully.
12. $\mathcal{M}$ outputs a guess: $\mathcal{S}$ aborts with failure.

Analysis of event $DL \wedge M^*$. $\mathcal{S}$'s simulation of $\mathcal{M}$'s environment is perfect except with negligible probability. With probability at least $\frac{1}{\mathbf{n}}$, $\mathcal{S}$ assigns the public key V to an honest party $\hat{B}$ for whom $\mathcal{M}$ will query $\mathcal{H}_1(*, b)$ without first issuing a *StaticKeyReveal*($\hat{B}$) query. In this case $\mathcal{S}$ is successful as described in Step 6 and the abortions as in Steps 7 and 12 do not occur. Hence if event $DL \wedge M^*$ occurs with probability p_{DL}, then $\mathcal{S}$ is successful with probability $\mathcal{P}(S)$ that is bounded by

$$\mathcal{P}(\mathcal{S}) \geq \frac{1}{\mathbf{n}} p_{DL}. \tag{3.1}$$

Event $\overline{DL}$ Let TM be the event "the test session has a matching session owned by an honest party". Event $\overline{DL} \wedge M^*$ is further subdivided into the following complementary events: (i) $T_m = (\overline{DL} \wedge M^* \wedge TM)$ and (ii) $\overline{T_m} = (\overline{DL} \wedge M^* \wedge \overline{TM})$. Let $p_{\tilde{DL}} = \mathcal{P}(\overline{DL} \wedge M^*)$, $p_m = \mathcal{P}(T_m)$, and $p_{\tilde{m}} = \mathcal{P}(\overline{T_m})$. Since T_m and $\overline{T_m}$ are complementary $p_{\tilde{DL}} = p_m + p_{\tilde{m}}$. Therefore, if event $\overline{DL} \wedge M^*$ occurs with non-negligible probability, then either T_m or $\overline{T_m}$ occurs with non-negligible probability. Events T_m and $\overline{T_m}$ are next considered separately.

Simulation in event T_m Suppose that event T_m occurs with non-negligible probability. In this case $\mathcal{S}$ establishes $\mathbf{n}$ honest parties that are assigned random static key pairs, and randomly selects two integers $i, j \in_R [1, \ldots, \mathbf{ns}]$. The i'th and the j'th sessions created will be called s^U and s^V, respectively. The ephemeral private key of s^U is denoted by $\tilde{u}$ and the ephemeral private keys of s^V is denoted by $\tilde{v}$. The simulation of $\mathcal{M}$'s environment proceeds as follows:

1. *Send*($\hat{A}, \hat{B}$): $\mathcal{S}$ executes Step 1 of the protocol. However, if the session being created is s^U or s^V, then $\mathcal{S}$ deviates by setting the ephemeral public key X to be U or V, respectively, thereby defining $\mathcal{H}_1(\tilde{u}, a) = \log_g U$ or $\mathcal{H}_1(\tilde{v}, a) = \log_g V$. Note that in this case $\mathcal{S}$ cannot respond to either $\mathcal{H}_1(\tilde{u}, a) = \log_g U$ or $\mathcal{H}_1(\tilde{v}, a) = \log_g V$.

2. *Send*($\hat{B}, \hat{A}, X$): $\mathcal{S}$ executes Step 2 of the protocol. However, if the session being created is s^U or s^V, $\mathcal{S}$ deviates by setting the ephemeral public key Y to be U or V, respectively, and setting $\sigma = \xi(XA^{\mathrm{D}}, YB^{\mathrm{E}})$.

3. *Send*($\hat{A}, \hat{B}, X, Y$): $\mathcal{S}$ executes Step 3 of the protocol. However, if $X \in \{U, V\}$ then $\mathcal{S}$ deviates by setting $\sigma = \xi(XA^{\mathrm{D}}, YB^{\mathrm{E}})$.

4. *EphemeralKeyReveal*(s): $\mathcal{S}$ responds to the query faithfully.

5. *SessionKeyReveal*(s): $\mathcal{S}$ responds to the query faithfully.

6. *StaticKeyReveal*($\hat{A}$): $\mathcal{S}$ responds to the query faithfully.

7. *Establish*($\hat{M}$): $\mathcal{S}$ responds to the query faithfully.

8. $\mathcal{H}_1(\tilde{x}, a)$: $\mathcal{S}$ simulates a random oracle in the usual way except if $\hat{A}$ owns s^U and $\tilde{x} = \tilde{u}$ or if $\hat{A}$ owns s^V and $\tilde{x} = \tilde{v}$, in which case $\mathcal{S}$ aborts with failure.

9. $\mathcal{H}_2(*)$: $\mathcal{S}$ simulates a random oracle in the usual way.

10. $\mathcal{H}(\sigma, X, Y, \hat{A}, \hat{B})$:

    (a) If $\{X, Y\} = \{U, V\}$ and $\mathrm{DDH}(XA^{\mathrm{D}}, YB^{\mathrm{E}}, \sigma) = 1$, then $\mathcal{S}$ aborts $\mathcal{M}$ and outputs $\mathrm{CDH}(U, V) = \sigma g^{-ab\mathrm{ED}} X^{-b\mathrm{E}} Y^{-a\mathrm{D}}$.

    (b) If $X \in \{U, V\}$ and $\sigma \neq \xi(XA^{\mathrm{D}}, YB^{\mathrm{E}})$, then $\mathcal{S}$ obtains $\tau = \mathrm{DDH}(XA^{\mathrm{D}}, YB^{\mathrm{E}}, \sigma)$.

        i. If $\tau = 1$, then $\mathcal{S}$ returns $\mathcal{H}\left(\xi(XA^{\mathrm{D}}, YB^{\mathrm{E}}), X, Y, \hat{A}, \hat{B}\right)$.

        ii. If $\tau = 0$, then $\mathcal{S}$ simulates a random oracle in the usual way.

    (c) If $Y \in \{U, V\}$ and $\sigma \neq \xi(XA^{\mathrm{D}}, YB^{\mathrm{E}})$, then $\mathcal{S}$ obtains $\tau = \mathrm{DDH}(XA^{\mathrm{D}}, YB^{\mathrm{E}}, \sigma)$.

        i. If $\tau = 1$, then $\mathcal{S}$ returns $\mathcal{H}\left(\xi(XA^{\mathrm{D}}, YB^{\mathrm{E}}), X, Y, \hat{A}, \hat{B}\right)$.

        ii. If $\tau = 0$, then $\mathcal{S}$ simulates a random oracle in the usual way.

    (d) $\mathcal{S}$ simulates a random oracle in the usual way.

11. *Test*(s^t): If s^U and s^V are non-matching or if s^t is neither s^U nor s^V, then $\mathcal{S}$ aborts; otherwise responds to the query faithfully.

12. $\mathcal{M}$ outputs a guess: $\mathcal{S}$ aborts with failure.

Analysis of event $T_m \wedge \overline{DL} \wedge M^*$. $\mathcal{S}$'s simulation of $\mathcal{M}$'s environment is perfect except with negligible probability. The probability that $\mathcal{M}$ selects s^U and s^V as the test session and its matching is at least $\frac{2}{(\mathbf{ns})^2}$. Suppose that this is the case, so $\mathcal{S}$ does not abort as in Step 11, and suppose that event T_m occurs. Without loss of generality, let $\mathrm{s}^t = \mathrm{s}^U = (\hat{A}, \hat{B}, U, V)$. Since $\tilde{u}$ is used only in the test session, $\mathcal{M}$ must obtain it via an *EphemeralKeyReveal* query before making an $\mathcal{H}_1$ query that includes $\tilde{u}$. Similarly, $\mathcal{M}$ must obtain $\tilde{v}$ from the matching session via an *EphemeralKeyReveal* query before making an $\mathcal{H}_1$ query that includes $\tilde{v}$. Under event $\overline{DL}$, the adversary first issues a *StaticKeyReveal* query to a party before making an $\mathcal{H}_1$ query that includes that party's static private key. Since the test session is fresh, $\mathcal{M}$ can query for at most one value in each of the pairs $(\tilde{u}, a)$ and $(\tilde{v}, b)$; hence $\mathcal{S}$ does

not abort as described in Step 8. Under event M^*, except with negligible probability of guessing $\xi(UA^{\mathrm{D}}, VB^{\mathrm{E}})$, $\mathcal{S}$ is successful as described in Step 10(a) and does not abort as in Step 12. Therefore if event T_m occurs, then the success probability of $\mathcal{S}$ is bounded by

$$\mathcal{P}(\mathcal{S}) \geq \frac{2}{(\mathbf{ns})^2} p_m. \tag{3.2}$$

Simulation in event $\overline{T_m}$. Suppose that event $\overline{T_m}$ occurs with non-negligible probability. Recall that event $\overline{T_m}$ implies that no honest party owns a session matching to the test session. In this case $\mathcal{S}$ prepares $\mathbf{n}$ parties. One party, called $\hat{V}$, is selected at random and assigned static public key V and $\mathcal{S}$ represents $\hat{V}$'s static private key by $\nu \in_R \mathbb{Z}_q$. The remaining $\mathbf{n} - 1$ parties are assigned random static key pairs. Furthermore, $\mathcal{S}$ randomly selects an integer $i \in_R [1, \ldots, \mathbf{ns}]$. The i'th session created will be called S^U and S^U's ephemeral private key will be denoted by $\tilde{u}$. The simulation of $\mathcal{M}$'s environment proceeds as follows:

1. *Send*$(\hat{A}, \hat{B})$: $\mathcal{S}$ executes Step 1 of the protocol. However, if the session being created is S^U, then $\mathcal{S}$ deviates by setting the ephemeral public key X to be U.
2. *Send*$(\hat{B}, \hat{A}, X)$: $\mathcal{S}$ executes Step 2 of the protocol. However, if the session being created is S^U, $\mathcal{S}$ deviates by setting the ephemeral public key Y to be U. In addition if $\hat{B} = \hat{V}$ or $Y = V$, then $\mathcal{S}$ sets $\sigma = \xi(XA^{\mathrm{D}}, YB^{\mathrm{E}})$.
3. *Send*$(\hat{A}, \hat{B}, X, Y)$: $\mathcal{S}$ executes Step 3 of the protocol. However, if $\hat{A} = \hat{V}$ or $X = U$, then $\mathcal{S}$ deviates by setting $\sigma = \xi(XA^{\mathrm{D}}, YB^{\mathrm{E}})$.
4. *EphemeralKeyReveal*(S): $\mathcal{S}$ responds to the query faithfully.
5. *SessionKeyReveal*(S): $\mathcal{S}$ responds to the query faithfully.
6. *StaticKeyReveal*$(\hat{A})$: $\mathcal{S}$ responds to the query faithfully, unless $\hat{A} = \hat{V}$ in which case $\mathcal{S}$ aborts with failure.
7. *Establish*$(\hat{M})$: $\mathcal{S}$ responds to the query faithfully.
8. $\mathcal{H}_1(\tilde{x}, a)$: $\mathcal{S}$ simulates a random oracle in the usual way except if $\hat{A}$ owns S^U and $\tilde{x} = \tilde{u}$, in which case $\mathcal{S}$ aborts with failure.
9. $\mathcal{H}_2(*)$: $\mathcal{S}$ simulates a random oracle in the usual way.
10. $\mathcal{H}(\sigma, X, Y, \hat{A}, \hat{B})$:
    (a) If $X = U$ and $\hat{B} = \hat{V}$, then $\mathcal{S}$ obtains $\tau = \mathrm{DDH}(XA^{\mathrm{D}}, YB^{\mathrm{E}}, \sigma)$.

i. If $\tau = 1$, then $\mathcal{S}$ computes $\Pi = \sigma Y^{-a\mathrm{D}} V^{-a\mathrm{DE}} = g^{uv\mathrm{E}+uy}$.

ii. If $\tau = 0$, then $\mathcal{S}$ simulates a random oracle in the usual way.

(b) If $Y = U$ and $\hat{A} = \hat{V}$, then $\mathcal{S}$ obtains $\tau = \mathrm{DDH}(XA^{\mathrm{D}}, YB^{\mathrm{E}}, \sigma)$

i. If $\tau = 1$, then $\mathcal{S}$ computes $\Pi = \sigma X^{-b\mathrm{E}} V^{-b\mathrm{DE}} = g^{uv\mathrm{D}+uy}$.

ii. If $\tau = 0$, then $\mathcal{S}$ simulates a random oracle in the usual way.

(c) If $\sigma \neq \xi(XA^{\mathrm{D}}, YB^{\mathrm{E}})$ and either $U \in \{X, Y\}$ or $\hat{V} \in \{\hat{A}, \hat{B}\}$, then $\mathcal{S}$ obtains $\tau = \mathrm{DDH}(XA^{\mathrm{D}}, YB^{\mathrm{E}}, \sigma)$.

i. If $\tau = 1$, then $\mathcal{S}$ returns $\mathcal{H}\left(\xi(XA^{\mathrm{D}}, YB^{\mathrm{E}}), X, Y, \hat{A}, \hat{B}\right)$.

ii. If $\tau = 0$, then $\mathcal{S}$ simulates a random oracle in the usual way.

(d) $\mathcal{S}$ simulates a random oracle in the usual way.

11. *Test*(s^t): If $\mathrm{s}^t \neq \mathrm{s}^U$ or the peer of s^t is not $\hat{V}$, then $\mathcal{S}$ aborts with failure; otherwise responds to the query faithfully.

12. $\mathcal{M}$ outputs a guess: $\mathcal{S}$ aborts with failure.

Analysis of event $\overline{T_m} \wedge \overline{DL} \wedge M^*$. The simulation of $\mathcal{M}$'s environment is perfect except with negligible probability. The probability that the test session has peer $\hat{V}$ and outgoing ephemeral public key U is at least $\frac{1}{\mathbf{n^2 s}}$. Suppose that this is indeed the case, so $\mathcal{S}$ does not abort as in Step 11, and suppose that event $\overline{T_m}$ occurs. Since $\tilde{u}$ is used only in the test session, $\mathcal{M}$ must obtain it via an *EphemeralKeyReveal* query before making an $\mathcal{H}_1$ query that includes $\tilde{u}$. Under event $\overline{DL}$, the adversary first issues a *StaticKeyReveal* query to a party before making an $\mathcal{H}_1$ query that includes that party's static private key. Since the test session is fresh, and s^t has no matching session a successful $\mathcal{M}$ does not query for $\tilde{x}$; hence $\mathcal{S}$ does not abort as described in Steps 6 and 8.

Without loss of generality let Y denote the incoming ephemeral public key selected by $\mathcal{M}$ for the test session $\mathrm{s}^t = (\hat{A}, \hat{V}, U, Y)$. Under event M^*, $\mathcal{M}$ queries $\mathcal{H}$ with $(\sigma, U, Y, \hat{A}, \hat{V})$ where $\mathrm{DDH}(UA^{\mathrm{D}}, YV^{\mathrm{E}}, \sigma) = 1$, in which case as described in Step 10(a) $\mathcal{S}$ computes

$$\Pi = \sigma Y^{-a\mathrm{D}} V^{-a\mathrm{DE}} = g^{uv\mathrm{E}+uy}.$$

Without the knowledge of $y = \log_g Y$, $\mathcal{S}$ is unable to compute $\mathrm{CDH}(U, V)$. Following the Forking Lemma [61, Lemma 2] approach[1], $\mathcal{S}$ runs $\mathcal{M}$ on the same input

[1]The Forking Lemma was introduced by Pointcheval and Stern to analyze the security of ElGamal-like signature schemes in the random oracle model. In that setting it is assumed that a forger $\mathcal{M}$ can,

and the same coin flips but with carefully modified answers to the $\mathcal{H}_2$ queries. Note that $\mathcal{M}$ must have queried $\mathcal{H}_2$ with $(Y, \hat{A}, \hat{B})$ in its first run, because otherwise $\mathcal{M}$ would be unable to compute σ except with negligible probability. For the second run of $\mathcal{M}$, $\mathcal{S}$ responds to $\mathcal{H}_2(Y, \hat{A}, \hat{B})$ with a value $\mathrm{E}' \neq \mathrm{E}$ selected uniformly at random. Another way of describing the second run is: $\mathcal{M}$ is rewound to the point where $\mathcal{M}$ queries $\mathcal{H}_2$ with $(Y, \hat{A}, \hat{B})$ and the query is answered with a random value E' different from E. If $\mathcal{M}$ succeeds in the second run, in Step 10(a) $\mathcal{S}$ computes

$$\Pi' = \sigma' Y^{-a\mathrm{D}'} V^{-a\mathrm{D}'\mathrm{E}'} = g^{uv\mathrm{E}'+uy}$$

and thereafter obtains

$$\mathrm{CDH}(U, V) = \left(\frac{\Pi}{\Pi'}\right)^{(\mathrm{E}-\mathrm{E}')^{-1}}.$$

The forking is at the expense of introducing a wider gap in the reduction. The success probability of $\mathcal{S}$, excluding negligible terms, is

$$\mathcal{P}(\mathcal{S}) \geq \frac{1}{\mathbf{s}} \frac{1}{\mathbf{n}^2} \frac{C}{h_2} p_{\tilde{m}} \tag{3.3}$$

where C is a constant arising from the use of the Forking Lemma[2]

Overall analysis. Suppose that event M occurs. Combining Equations (3.1), (3.2) and (3.3), the success probability of $\mathcal{S}$ is

$$\mathcal{P}(\mathcal{S}) \geq \max\left\{ \frac{1}{\mathbf{n}(\lambda)} p_{DL}(\lambda), \frac{2}{(\mathbf{n}(\lambda)\mathbf{s}(\lambda))^2} p_m(\lambda), \frac{C}{\mathbf{s}(\lambda)\mathbf{n}(\lambda)^2 h_2(\lambda)} p_{\tilde{m}}(\lambda) \right\}, \tag{3.4}$$

which is non-negligible in λ.

The simulation requires $\mathcal{S}$ to perform group exponentiations, access the DDH oracle, and simulate random oracles. Since $q = \Theta(2^\lambda)$, a group exponentiation takes time $\mathcal{T}_\mathcal{G} = \mathcal{O}(\lambda)$ group multiplications. Assume that a DDH oracle call takes time $\mathcal{T}_{\mathrm{DDH}} = \mathcal{O}(\lambda)$. Responding to an $\mathcal{H}$ query takes time $\mathcal{T}_\mathcal{H} = \mathcal{O}(\lambda)$; similarly, responding to $\mathcal{H}_1$ and $\mathcal{H}_2$ queries takes time $\mathcal{T}_{\mathcal{H}_1}(\lambda)$ and $\mathcal{T}_{\mathcal{H}_2}(\lambda)$. Taking the largest

with non-negligible probability, create a signature on a message m that $\mathcal{M}$ did not previously give to the signing oracle. The lemma states that if the forger is replayed with the same input and coin flips, and the same responses to its signing and random oracle queries up to the point when m is queried to the random oracle (in which case a different random hash value is returned), then the forger will succeed in creating a forgery for that same message m with non-negligible probability. The Forking Lemma approach has been used to prove security of different kinds of protocols. Here it is used essentially the same way as in Krawcyzk's [41, §4.2] proof of the XCR signature scheme.

[2]The constant C in Pointcheval and Stern's version of the lemma is 84480^{-1}. Its value has not been worked out in Theorem 3.4.1's (and Krawzcyk's) use of the Forking Lemma.

times from among all simulations for answering $\mathcal{M}$'s queries, the running time of $\mathcal{S}$ is bounded by

$$\mathcal{T}_{\mathcal{S}} \leq (\mathcal{T}_{3\mathcal{G}} + (\mathcal{T}_{\mathrm{DDH}} + 2\mathcal{T}_{\mathcal{G}} + \mathcal{T}_{\mathcal{H}}) + (\mathcal{T}_{\mathcal{G}} + \mathcal{T}_{\mathcal{H}_1}) + \mathcal{T}_{\mathcal{H}_2})\mathcal{T}_{\mathcal{M}}. \tag{3.5}$$

Thus, if $\mathcal{M}$ is polynomially bounded, then there is an algorithm $\mathcal{S}$ that succeeds in solving the GDH problem in $\mathcal{G}$ with non-negligible probability. Furthermore $\mathcal{S}$ runs in polynomial time, contradicting the GDH assumption in $\mathcal{G}$. This concludes the proof of Theorem 3.4.1. □

3.5 One-pass CMQV

In a nutshell, one-pass CMQV is two-pass CMQV, where the responder's ephemeral public key Y is the identity element in the group. To that end there is no need to include Y in the key derivation.

3.5.1 Protocol description

As before, in the protocol description, λ is the security parameter; $\mathcal{H}_1 : \{0,1\}^{\lambda} \times \mathbb{Z}_q^* \to \mathbb{Z}_q^*$, $\mathcal{H}_2 : \{0,1\}^* \to \mathbb{Z}_q$ and $\mathcal{H} : \{0,1\}^* \to \{0,1\}^{\lambda}$ are random oracles; and $\hat{A}$ and $\hat{B}$ are two parties with valid public-private key pairs (A, a) and (B, b), respectively. The one-pass CMQV protocol is formally given in the following definition:

Definition 3.5.1 (one-pass CMQV) *The protocol proceeds as follows:*

1. *Upon activation $(\hat{A}, \hat{B})$, party $\hat{A}$ (the initiator) performs the steps:*
   (a) *Select an ephemeral private key $\tilde{x} \in_R \{0,1\}^{\lambda}$.*
   (b) *Compute the ephemeral public key $X = g^{\mathcal{H}_1(\tilde{x}, a)}$.*
   (c) *Compute $\mathsf{D} = \mathcal{H}_2(X, \hat{A}, \hat{B})$ and $\sigma = B^{\mathcal{H}_1(\tilde{x}, a) + \mathsf{D}a}$.*
   (d) *Compute $\kappa = \mathcal{H}(\sigma, X, \hat{A}, \hat{B})$ and destroy $\tilde{x}$ and σ.*
   (e) *Send $(\hat{B}, \hat{A}, X)$ to $\hat{B}$ and complete session $\mathsf{S} = (\hat{A}, \hat{B}, X)$ with session key κ.*
2. *Upon activation $(\hat{B}, \hat{A}, X)$, party $\hat{B}$ (the responder) performs the steps:*
   (a) *Verify that $X \in \mathcal{G}^*$.*
   (b) *Compute $\mathsf{D} = \mathcal{H}_2(X, \hat{A}, \hat{B})$ and $\sigma = (XA^{\mathsf{D}})^b$.*
   (c) *Compute $\kappa = \mathcal{H}(\sigma, X, \hat{A}, \hat{B})$ and destroy σ.*

(d) Complete session $\mathrm{s} = (\hat{B}, \hat{A}, X)$ *with session key* κ.

If any verification fails, then the party erases all session specific information from its memory and aborts the session.

3.5.2 Model modifications

Even though the definition of a secure protocol (Definition 2.5.2) does not depend on the number of protocol flows, the fresh session definition has to be modified to fit the needs of a one-pass protocol. In particular, one-pass protocols cannot achieve wFS since an adversary can compute the test session key by learning the static private key of the responder.

Definition 3.5.2 (one-pass fresh session) *Let* s *be the session identifier of a completed session, owned by an honest party* $\hat{A}$ *with intended peer* $\hat{B}$*, who is also honest. Let* s^* *be the session identifier of the matching session of* s*, if it exists. Define* s *to be* fresh *if none of the following conditions hold:*

(i) $\mathcal{M}$ *issues a* SessionKeyReveal(s) *query or a* SessionKeyReveal(s^*) *query (if* s^* *exists);*

(ii) if $\hat{A}$ *is the initiator, then* $\mathcal{M}$ *makes either of the following queries:*

- *both* StaticKeyReveal($\hat{A}$) *and* EphemeralKeyReveal(s)*, or*
- StaticKeyReveal($\hat{B}$)*;*

(iii) if $\hat{A}$ *is the responder, then* $\mathcal{M}$ *makes either of the following queries*

- StaticKeyReveal($\hat{A}$) *or*
- StaticKeyReveal($\hat{B}$)*.*

By replaying messages from $\hat{A}$ to $\hat{B}$ an adversary $\mathcal{M}$ could force multiple sessions owned by $\hat{B}$ sharing the same session key κ. Let S_κ be the set of sessions owned by $\hat{B}$ with the same session key κ. Since all sessions in S_κ have the same session identifiers, $\mathcal{M}$ cannot compromise a single session in S_κ without compromising all sessions in S_κ. Therefore, the definition of session identifier accounts for replay attacks.

3.5.3 Security argument

The security argument for one-pass CMQV is very similar to the security argument for two-pass CMQV (§3.4.4).

Theorem 3.5.1 *If $\mathcal{H}_1$,$\mathcal{H}_2$ and $\mathcal{H}$ are random oracles, and $\mathcal{G}$ is a group where the GDH assumption holds, then one-pass CMQV is eCK secure.*

Proof: Verifying condition 1 of Definition 2.5.2 is straightforward; it remains to verify condition 2.

Let λ denote the security parameter, whence $q = |\mathcal{G}| = \Theta(2^\lambda)$. Let $\mathcal{M}$ be a polynomially (in λ) bounded one-pass CMQV adversary. The adversary $\mathcal{M}$ is said to be successful (event M) with non-negligible probability if $\mathcal{M}$ wins the distinguishing game described in §2.5 with probability $\frac{1}{2} + p(\lambda)$, where $p(\lambda)$ is non-negligible. Assume that $\mathcal{M}$ operates in an environment that involves at most $\mathbf{n}(\lambda)$ parties, and within a party $\mathcal{M}$ activates at most $\mathbf{s}(\lambda)$ sessions as the initiator within a party, and makes at most $h_1(\lambda), h_2(\lambda)$ and $h(\lambda)$ queries to oracles $\mathcal{H}_1$,$\mathcal{H}_2$ and $\mathcal{H}$, respectively; and terminates after time at most $\mathcal{T}_\mathcal{M}$. Let the test session be $\mathrm{s}^t = (\hat{A}, \hat{B}, X)$ and let H denote the event that $\mathcal{M}$ queries $\mathcal{H}$ with $(\sigma, X, \hat{A}, \hat{B})$, where $\sigma = \mathrm{CDH}(XA^{\mathrm{D}}, B)$. Let $\overline{H}$ be the complement of H and s^* be any completed session owned by an honest party, such that s^t and s^* are non-matching. Since s^t and s^* are non-matching, the input to the key derivation function $\mathcal{H}$ are different for s^t and s^*. And since $\mathcal{H}$ is a random oracle it follows that $\mathcal{M}$ cannot obtain any information about the test session key from the session key of non-matching sessions. Hence $\mathcal{P}(M \wedge \overline{H}) \leq \frac{1}{2}$ and

$$\mathcal{P}(M) = \mathcal{P}(M \wedge \overline{H}) + \mathcal{P}(M \wedge H) \leq \frac{1}{2} + \mathcal{P}(M \wedge H),$$

whence $\mathcal{P}(M \wedge H) \geq p$; henceforth the event $M \wedge H$ is denoted by M^*.

Following the standard approach, such an adversary $\mathcal{M}$ is used to construct a GDH solver $\mathcal{S}$ that succeeds with non-negligible probability. Let $\xi : \mathcal{G} \times \mathcal{G} \rightarrow \mathcal{G}$ be a random function known only to $\mathcal{S}$, such that $\xi(X, Y) = \xi(Y, X)$. The algorithm $\mathcal{S}$ will use ξ to simulate $\mathrm{CDH}(X, Y)$ when $\mathcal{S}$ may not know $\log_g(X)$ or $\log_g(Y)$. Let the input to the GDH challenge be (U, V) and consider the following complementary events:

DL. There exists an honest party $\hat{B}$ such that $\mathcal{M}$, during its execution, queries $\mathcal{H}_1$ with $(*, b)$, before issuing a *StaticKeyReveal*$(\hat{B})$ query. (Note that $\mathcal{M}$ does not necessarily make a *StaticKeyReveal*$(\hat{B})$ query.)

$\overline{DL}$. During its execution, for every honest party $\hat{B}$ for which $\mathcal{M}$ queries $\mathcal{H}_1$ with $(*, b)$, $\mathcal{M}$ issued *StaticKeyReveal*($\hat{B}$) before the first $(*, b)$ query to $\mathcal{H}_1$.

If $\mathcal{M}$ succeeds with non-negligible probability, and hence $\mathcal{P}(M^*) \geq p$, it must be the case that either event $DL \wedge M^*$ or event $\overline{DL} \wedge M^*$ occurs with non-negligible probability. These events are considered separately.

Simulation in event DL Suppose that event $DL \wedge M^*$ occurs with non-negligible probability. In this case $\mathcal{S}$ prepares $\mathbf{n}$ parties. One party, called $\hat{V}$, is selected at random and assigned static public key V; $\mathcal{S}$ represents $\hat{V}$'s static private key by $\nu \in_R \mathbb{Z}_q$. The remaining $\mathbf{n}-1$ parties are assigned random static key pairs. The adversary $\mathcal{M}$ is initiated on this set of parties and the simulation of $\mathcal{M}$'s environment proceeds as follows:

1. *Send*($\hat{A}, \hat{B}$): $\mathcal{S}$ executes Step 1 of the protocol. However, if $\hat{A} = \hat{V}$, then $\mathcal{S}$ deviates by setting $\sigma = \xi(XA^{\mathrm{D}}, B)$.
2. *Send*($\hat{B}, \hat{A}, X$): $\mathcal{S}$ executes Step 2 of the protocol. However, if $\hat{B} = \hat{V}$, then $\mathcal{S}$ sets $\sigma = \xi(XA^{\mathrm{D}}, B)$.
3. *EphemeralKeyReveal*(s): $\mathcal{S}$ responds to the query faithfully.
4. *SessionKeyReveal*(s): $\mathcal{S}$ responds to the query faithfully.
5. *StaticKeyReveal*($\hat{A}$): $\mathcal{S}$ responds to the query faithfully, unless $\hat{A} = \hat{V}$ in which case $\mathcal{S}$ aborts with failure.
6. *Establish*($\hat{M}$): $\mathcal{S}$ responds to the query faithfully.
7. $\mathcal{H}_1(s, c)$: $\mathcal{S}$ checks if $g^c = V$; if the equation holds, then $\mathcal{S}$ stops $\mathcal{M}$ and outputs $\mathrm{GDH}(U, V) = V^c$. In all other cases $\mathcal{S}$ simulates a random oracle in the usual way.
8. $\mathcal{H}_2(*)$: $\mathcal{S}$ simulates a random oracle in the usual way.
9. $\mathcal{H}(\sigma, X, \hat{A}, \hat{B})$:
   (a) If $\hat{V} \in \{\hat{A}, \hat{B}\}$ and $\sigma \neq \xi(XA^{\mathrm{D}}, B)$, then $\mathcal{S}$ obtains $\tau = \mathrm{DDH}(XA^{\mathrm{D}}, B, \sigma)$.
      i. If $\tau = 1$, then $\mathcal{S}$ returns $\mathcal{H}\left(\xi(XA^{\mathrm{D}}, B), X, \hat{A}, \hat{B}\right)$.
      ii. If $\tau = 0$, then $\mathcal{S}$ simulates a random oracle in the usual way.
   (b) $\mathcal{S}$ simulates a random oracle in the usual way.

10. *Test*(s): $\mathcal{S}$ responds to the query faithfully.

11. $\mathcal{M}$ outputs a guess: $\mathcal{S}$ aborts with failure.

Analysis of event $DL \wedge M^*$. $\mathcal{S}$'s simulation of $\mathcal{M}$'s environment is perfect except with negligible probability. With probability at least $\frac{1}{\mathbf{n}}$, $\mathcal{S}$ assigns the public key V to an honest party $\hat{B}$ for whom $\mathcal{M}$ will query $\mathcal{H}_1(*, b)$ without first issuing a *StaticKeyReveal*($\hat{B}$) query. In this case $\mathcal{S}$ is successful as described in Step 6 and the abortions as in Steps 7 and 12 do not occur. Hence if event $DL \wedge M^*$ occurs with probability p_{DL}, then $\mathcal{S}$ is successful with probability $\mathcal{P}(S)$ that is bounded by

$$\mathcal{P}(\mathcal{S}) \geq \frac{1}{\mathbf{n}} p_{DL}. \tag{3.6}$$

Event $\overline{DL}$ Let T_m be the event "the test session has a matching session owned by an honest party or the test session owner is also the session initiator". Event $\overline{DL} \wedge M^*$ is further subdivided into the following complementary events: (i) $T_m = (\overline{DL} \wedge M^* \wedge T_m)$ and (ii) $\overline{T_m} = (\overline{DL} \wedge M^* \wedge \overline{TM})$. Let $p_{\tilde{DL}} = \mathcal{P}(\overline{DL} \wedge M^*)$, $p_m = \mathcal{P}(T_m)$, and $p_{\tilde{m}} = \mathcal{P}(\overline{T_m})$. Since T_m and $\overline{T_m}$ are complementary, $p_{\tilde{DL}} = p_m + p_{\tilde{m}}$. Therefore, if event $\overline{DL} \wedge M^*$ occurs with non-negligible probability, then either T_m or $\overline{T_m}$ occurs with non-negligible probability. Events T_m and $\overline{T_m}$ are next considered separately.

Simulation in event T_m. Suppose that event T_m occurs with non-negligible probability. In this case $\mathcal{S}$ establishes $\mathbf{n}$ parties. One party, called $\hat{V}$, is selected at random and assigned static public key V; $\mathcal{S}$ represents $\hat{V}$'s static private key by $\nu \in_R \mathbb{Z}_q$. The remaining $\mathbf{n} - 1$ parties are assigned random static key pairs. Furthermore, $\mathcal{S}$ randomly selects an integer $i \in_R [1, \ldots, \mathbf{ns}]$. The i'th session created will be called s^U and s^U's ephemeral private key will be denoted by $\tilde{u}$. The simulation of $\mathcal{M}$'s environment proceeds as follows:

1. *Send*($\hat{A}, \hat{B}$): $\mathcal{S}$ executes Step 1 of the protocol. However, if the session being created is s^U, $\mathcal{S}$ deviates by setting the ephemeral public key X to be U. In addition, if $X = U$ or $\hat{A} = \hat{V}$, then $\mathcal{S}$ sets $\sigma = \xi(XA^{\mathrm{D}}, B)$.

2. *Send*($\hat{B}, \hat{A}, X$): $\mathcal{S}$ executes Step 2 of the protocol. However, if $\hat{B} = \hat{V}$, then $\mathcal{S}$ deviates by setting $\sigma = \xi(XA^{\mathrm{D}}, B)$.

3. *EphemeralKeyReveal*(s): $\mathcal{S}$ responds to the query faithfully.

4. *SessionKeyReveal*(s): $\mathcal{S}$ responds to the query faithfully.

5. *StaticKeyReveal*($\hat{A}$): $\mathcal{S}$ responds to the query faithfully, unless $\hat{A} = \hat{V}$, in which case $\mathcal{S}$ aborts with failure.

6. *Establish*($\hat{M}$): $\mathcal{S}$ responds to the query faithfully.

7. $\mathcal{H}_1(\tilde{x}, a)$: $\mathcal{S}$ simulates a random oracle in the usual way except if $\hat{A}$ owns s^U and $\tilde{x} = \tilde{u}$ or if $\hat{A}$ owns s^V and $\tilde{x} = \tilde{v}$, in which case $\mathcal{S}$ aborts with failure.

8. $\mathcal{H}_2(*)$: $\mathcal{S}$ simulates a random oracle in the usual way.

9. $\mathcal{H}(\sigma, X, \hat{A}, \hat{B})$:

   (a) If $X = U$, $\hat{B} = \hat{V}$ and $\mathrm{DDH}(XA^{\mathrm{D}}, B, \sigma) = 1$, then $\mathcal{S}$ aborts $\mathcal{M}$ and outputs $\mathrm{CDH}(V) = \sigma V^{-a\mathrm{D}}$.

   (b) If $\sigma \neq \xi(XA^{\mathrm{D}}, Y)$ and either $\hat{V} \in \{\hat{A}, \hat{B}\}$ or $X = U$, then $\mathcal{S}$ obtains $\tau = \mathrm{DDH}(XA^{\mathrm{D}}, B, \sigma)$.

      i. If $\tau = 1$, then $\mathcal{S}$ returns $\mathcal{H}\left(\xi(XA^{\mathrm{D}}, Y), X, \hat{A}, \hat{B}\right)$.

      ii. If $\tau = 0$, then $\mathcal{S}$ simulates a random oracle in the usual way.

   (c) $\mathcal{S}$ simulates a random oracle in the usual way.

10. *Test*(s^t): If the peer of s^U is not $\hat{V}$ or if s^t is neither s^U nor the session matching to s^U, then $\mathcal{S}$ aborts; otherwise responds to the query faithfully.

11. $\mathcal{M}$ outputs a guess: $\mathcal{S}$ aborts with failure.

Analysis of event $T_m \wedge \overline{DL} \wedge M^*$. $\mathcal{S}$'s simulation of $\mathcal{M}$'s environment is perfect except with negligible probability. The probability that $\mathcal{M}$ selects $\hat{V}$ as s^U's peer and s^t is either s^U or its matching session is at least $\frac{2}{\mathbf{ns}^2}$. Suppose that this is the case, so $\mathcal{S}$ does not abort as in Step 10, and suppose that event T_m occurs. Without loss of generality, let $\mathrm{s}^U = (\hat{A}, \hat{V}, U)$. Since $\tilde{u}$ is used only in s^U, $\mathcal{M}$ must obtain $\tilde{u}$ via an *EphemeralKeyReveal* query before making an $\mathcal{H}_1$ query that includes $\tilde{u}$. Under event $\overline{DL}$, the adversary first issues a *StaticKeyReveal*($\hat{A}$) query before making an $\mathcal{H}_1$ query that includes $\tilde{x}$. Since the test session is fresh, $\mathcal{S}$ does not abort as described in Step 6 and 7. Under event M^*, except with negligible probability of guessing $\xi(UA^{\mathrm{D}}, V)$, $\mathcal{S}$ is successful as described in Step 10(a) and does not abort as in Step 11. Therefore if event T_m occurs, then the success probability of $\mathcal{S}$ is

$$\mathcal{P}(\mathcal{S}) \geq \frac{2}{\mathbf{ns}^2} p_m. \tag{3.7}$$

Simulation in event $\overline{T_m}$. Suppose that event $\overline{T_m}$ occurs with non-negligible probability, in which case no honest party owns a session matching to the test session and the test session owner is also the responder. In this case $\mathcal{S}$ prepares $\mathbf{n}$ parties. Two of these parties, denoted by $\hat{U}$ and $\hat{V}$, are selected uniformly at random and assigned static public keys U and V, respectively. The remaining $\mathbf{n}-2$ parties are assigned random static key pairs. The algorithm $\mathcal{S}$ will use $\upsilon \in_R \mathbb{Z}_q$ and $\nu \in_R \mathbb{Z}_q$, to represent the static private keys of $\hat{U}$ and $\hat{V}$, respectively. The simulation of $\mathcal{M}$'s environment proceeds as follows:

1. *Send*$(\hat{A}, \hat{B})$: $\mathcal{S}$ executes Step 1 of the protocol. However, if $\hat{A} \in \{\hat{U}, \hat{V}\}$, then $\mathcal{S}$ sets $\sigma = \xi(XA^{\mathrm{D}}, B)$.
2. *Send*$(\hat{B}, \hat{A}, X)$: $\mathcal{S}$ executes Step 2 of the protocol. However, if $\hat{B} \in \{\hat{U}, \hat{V}\}$, then $\mathcal{S}$ sets $\sigma = \xi(XA^{\mathrm{D}}, B)$.
3. *EphemeralKeyReveal*(s): $\mathcal{S}$ responds to the query faithfully.
4. *SessionKeyReveal*(s): $\mathcal{S}$ responds to the query faithfully.
5. *StaticKeyReveal*$(\hat{A})$: $\mathcal{S}$ responds to the query faithfully, unless $\hat{A} \in \{\hat{U}, \hat{V}\}$ in which case $\mathcal{S}$ aborts with failure.
6. *Establish*$(\hat{M})$: $\mathcal{S}$ responds to the query faithfully.
7. $\mathcal{H}_1(*)$: $\mathcal{S}$ simulates a random oracle in the usual way.
8. $\mathcal{H}_2(*)$: $\mathcal{S}$ simulates a random oracle in the usual way.
9. $\mathcal{H}(\sigma, X, \hat{A}, \hat{B})$:
   (a) If $\{\hat{A}, \hat{B}\} = \{\hat{U}, \hat{V}\}$ and $\mathrm{DDH}(XA^{\mathrm{D}}, B, \sigma) = 1$, then $\mathcal{S}$ records σ.
   (b) If $\sigma \neq \xi(XA^{\mathrm{D}}, B)$ and either $\hat{U} \in \{\hat{A}, \hat{B}\}$ or $\hat{V} \in \{\hat{A}, \hat{B}\}$, then $\mathcal{S}$ obtains $\tau = \mathrm{DDH}(XA^{\mathrm{D}}, B)$.
      i. If $\tau = 1$, then $\mathcal{S}$ returns $\mathcal{H}\left(\xi(XA^{\mathrm{D}}, B), X, \hat{A}, \hat{B}\right)$.
      ii. If $\tau = 0$, then $\mathcal{S}$ simulates a random oracle in the usual way.
   (c) $\mathcal{S}$ simulates a random oracle in the usual way.
10. *Test*(s^t): If the communicating partners of s^t are not $\hat{U}$ and $\hat{V}$, then $\mathcal{S}$ aborts with failure; otherwise responds to the query faithfully.
11. $\mathcal{M}$ outputs a guess: $\mathcal{S}$ aborts with failure.

Analysis of event $\overline{T_m} \wedge \overline{DL} \wedge M^*$. The simulation of $\mathcal{M}$'s environment is perfect except with negligible probability. The probability that $\hat{U}$ and $\hat{V}$ are the test session's communicating partners is at least $\frac{2}{\mathbf{n}^2}$. Suppose that this is indeed the case, so $\mathcal{S}$ does not abort as in Step 10, and suppose that event $\overline{T_m}$ occurs. Since the test session is fresh, and S^t has no matching session, then $\mathcal{S}$ does not abort as described in Steps 6 and 8.

Without loss of generality, let $\mathrm{S}^t = (\hat{U}, \hat{V}, Y)$, where Y denotes S^t's incoming ephemeral public key selected by $\mathcal{M}$. Under event M^*, except with negligible probability of guessing $\xi(YU^{\mathrm{D}}, V)$, $\mathcal{M}$ queries $\mathcal{H}$ with $(\sigma, Y, \hat{U}, \hat{V})$, where $\mathrm{DDH}(YU^{\mathrm{D}}, V, \sigma) = 1$, in which case as described in Step 10(a), $\mathcal{S}$ obtains

$$\sigma = g^{uv\mathrm{D}+vy}.$$

Without knowledge of $y = \log_g Y$, $\mathcal{S}$ is unable to compute $\mathrm{CDH}(U, V)$. Following the Forking Lemma [61] approach, $\mathcal{S}$ runs $\mathcal{M}$ on the same input and the same coin flips but with carefully modified answers to the $\mathcal{H}_2$ queries. Note that $\mathcal{M}$ must have queried $\mathcal{H}_2$ with $(Y, \hat{A}, \hat{B})$ in its first run, because otherwise $\mathcal{M}$ would be unable to compute σ except with negligible probability. For the second run of $\mathcal{M}$, $\mathcal{S}$ responds to $\mathcal{H}_2(Y, \hat{A}, \hat{B})$ with a value $\mathrm{D}' \neq \mathrm{D}$ selected uniformly at random. Another way of describing the second run is: $\mathcal{M}$ is rewound to the point where $\mathcal{M}$ queries $\mathcal{H}_2$ with $(Y, \hat{A}, \hat{B})$ and the query is answered with a random value D' different from D. If $\mathcal{M}$ succeeds in the second run, in Step 10(a) $\mathcal{S}$ obtains

$$\sigma' = g^{uv\mathrm{D}'+vy}$$

and thereafter obtains

$$\mathrm{CDH}(U, V) = \left(\frac{\sigma}{\sigma'}\right)^{(\mathrm{D}-\mathrm{D}')^{-1}}.$$

The forking is at the expense of introducing a wider gap in the reduction. The success probability of $\mathcal{S}$, excluding negligible terms, is

$$\mathcal{P}(\mathcal{S}) \geq \frac{1}{\mathbf{n}^2} \frac{C}{h_2} p_{\tilde{m}}, \tag{3.8}$$

where C is a constant arising from the Forking Lemma.

Overall analysis. Suppose that event M occurs. Combining Equations (3.6), (3.7) and (3.8), the success probability of $\mathcal{S}$ is

$$\mathcal{P}(\mathcal{S}) \geq \max\left\{\frac{1}{\mathbf{n}(\lambda)} p_{DL}(\lambda), \frac{2}{\mathbf{n}(\lambda)\mathbf{s}(\lambda)^2} p_m(\lambda), \frac{C}{\mathbf{n}(\lambda)^2 h_2(\lambda)} p_{\tilde{m}}(\lambda)\right\}, \tag{3.9}$$

which is non-negligible in λ.

The simulation requires $\mathcal{S}$ to perform group exponentiations, access the DDH oracle, and simulate random oracles. Since $q = \Theta(2^\lambda)$, a group exponentiation takes time $\mathcal{T}_\mathcal{G} = \mathcal{O}(\lambda)$ group multiplications. Assume that a DDH oracle call takes time $\mathcal{T}_{\mathrm{DDH}} = \mathcal{O}(\lambda)$. Responding to an $\mathcal{H}$ query takes time $\mathcal{T}_\mathcal{H} = \mathcal{O}(\lambda)$; similarly responding to $\mathcal{H}_1$ and $\mathcal{H}_2$ queries takes time $\mathcal{T}_{\mathcal{H}_1}(\lambda)$ and $\mathcal{T}_{\mathcal{H}_2}(\lambda)$, respectively. Taking the largest times from among all simulations for answering $\mathcal{M}$'s query, the running time of $\mathcal{S}$ is bounded by

$$\mathcal{T}_\mathcal{S} \leq (\mathcal{T}_{2\mathcal{G}} + (\mathcal{T}_{\mathrm{DDH}} + \mathcal{T}_\mathcal{G} + \mathcal{T}_\mathcal{H}) + (\mathcal{T}_\mathcal{G} + \mathcal{T}_{\mathcal{H}_1}) + \mathcal{T}_{\mathcal{H}_2})\mathcal{T}_\mathcal{M}. \tag{3.10}$$

Thus, if $\mathcal{M}$ is polynomially bounded, then there is an algorithm $\mathcal{S}$ that succeeds in solving the GDH problem in $\mathcal{G}$ with non-negligible probability. Furthermore $\mathcal{S}$ runs in polynomial time, contradicting the GDH assumption in $\mathcal{G}$. This concludes the proof of Theorem 3.5.1. □

3.6 Concluding remarks

On the positive side, the CMQV protocol is secure in the eCK model. Moreover it achieves the performance of the original MQV protocol, and has intuitive design principles and a relatively simple security proof. On the negative side, the reduction argument is not tight, in particular the Forking Lemma appears to be essential for the security argument. It remains to be seen if there exists a protocol that achieves the performance of MQV and at the same time enjoys a security reduction that is as tight as the security reduction for NAXOS.

Chapter 4

The UM protocol

4.1 Motivation

The Unified Model (UM) is a family of two-party Diffie-Hellman key agreement protocols that has been standardized in ANSI X9.42 [2], ANSI X9.63 [3], and more recently in the NIST SP800-56A [71]. The core protocol in the family is a two-pass protocol where each party contributes a static key pair and an ephemeral key pair which are then used to derive the secret session key. The family of protocols is called the 'unified model' because there are natural variants of the core protocol that are suitable in certain scenarios, for example in email where the receiver only contributes a static key pair. The UM protocol is believed to possess all important security attributes including key authentication and secrecy, resistance to unknown key-share attacks, forward secrecy, resistance to known-session key attacks, and resistance to leakage of ephemeral private keys, but is known to succumb to key-compromise impersonation attacks.

This chapter shall only consider the security of the core protocol which is called 'dhHybrid1' when the underlying group is a DSA-type group, and 'Full unified model' when the underlying group is an elliptic curve group [71]. More precisely, the focus will be on a three-pass variant that consists of the core protocol augmented with key confirmation as specified in the SP800-56A standard [71]. This variant is worthy of study because it possesses more security attributes than the other protocols in the unified model family and therefore is most likely to be deployed in applications that wish to be compliant with SP800-56A. Henceforth for simplicity, this protocol will be referred as *the* Unified Model (UM) protocol.

4.2 Previous work

Two previous papers [13] and [37] offered security proofs for variants of the UM protocol. A discussion of limitations of these results follows. For the remainder of this chapter it is assumed that parties can obtain authentic copies of each other's static public keys by exchanging certificates that have been issued by a trusted certifying authority (CA). Let MAC denote a message authentication code algorithm, and $\mathcal{H}$ and $\mathcal{H}_2$ denote independent hash functions.

The basic two-pass protocol upon which the UM protocol is built, is depicted in Figure 1.5. The communicating parties exchange static and ephemeral public keys and thereafter compute the session key $\kappa = \mathcal{H}(g^{xy}, g^{ab})$ by hashing the concatenation of the ephemeral Diffie-Hellman shared secret $\sigma_e = g^{xy}$ and the static Diffie-Hellman shared secret $\sigma_s = g^{ab}$. In [13] it was observed that this protocol is insecure under known-session key attacks. The attack highlights the importance of authenticating the exchanged ephemeral public keys. This led to Protocol 1, shown in Figure 4.1, and analyzed by Blake-Wilson, Johnson and Menezes [13].

$\hat{A}, a, A$		$\hat{B}, b, B$
$x, X, \sigma_e = Y^x$	$X \longrightarrow$	$y, Y, \sigma_e = X^y$
$\sigma_s = B^a$	$\longleftarrow Y, T_B$	$\sigma_s = A^b$
$\kappa_m = \mathcal{H}_2(\sigma_e, \sigma_s)$	$T_A \longrightarrow$	$\kappa_m = \mathcal{H}_2(\sigma_e, \sigma_s)$

$$T_A = \mathrm{MAC}_{\kappa_m}(2, \hat{A}, \hat{B}, X, Y) \qquad T_B = \mathrm{MAC}_{\kappa_m}(3, \hat{B}, \hat{A}, Y, X)$$

$$\kappa = \mathcal{H}(\sigma_e, \sigma_s)$$

Figure 4.1: Protocol 1 — the UM variant analyzed in [13].

In Protocol 1, the communicating parties also exchange key confirmation tags T_A, T_B computed using the MAC key $\kappa_m = \mathcal{H}_2(g^{xy}, g^{ab})$. If the tags verify, then the parties compute the session key $\kappa = \mathcal{H}(g^{xy}, g^{ab})$. Protocol 1 succumbs to a KCI attack since an adversary who learns $\hat{A}$'s static private key a can thereafter impersonate $\hat{B}$ (without knowing b) in a run of the protocol with $\hat{A}$. While KCI resilience is certainly desirable in practice, it is arguably not a fundamental security requirement of key agreement and a key agreement protocol should not be considered 'insecure' merely because it fails to be KCI resilient. Nevertheless, Protocol 1 appears to possess all the other desirable security attributes including key authentication and secrecy, resistance to unknown key-share attacks, forward secrecy, resistance

to known-session key attacks, and resistance to leakage of ephemeral private keys. In [13], the security model and definition developed by Bellare and Rogaway [9] for key agreement in the symmetric-key setting was adapted to the public-key setting. Protocol 1 was proven to meet this security definition in the random oracle model assuming that the CDH problem in $\mathcal{G}$ is intractable and that the MAC scheme is secure. However, the security model and result in [13] have the following shortcomings:

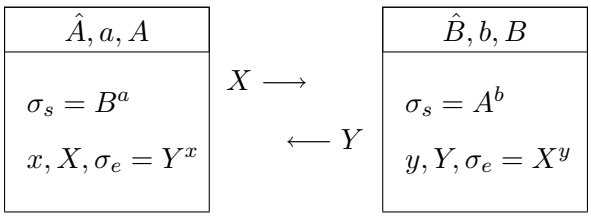
(i) The security model does not incorporate forward secrecy.

(ii) While the adversary is allowed to learn a party's static private key and thereafter impersonate the party, the security proof does not permit the adversary to replace that party's key pair with a key pair of its own choosing. Hence the security proof does not rule out 'malicious insider' attacks such as Kaliski's on-line attack [18]. (However, the security proof in [13] can be modified to rule out malicious insider attacks by invoking the stronger GDH assumption.)

(iii) The adversary is not allowed to learn any ephemeral private keys. More generally, the adversary is not allowed to learn any session-specific secret information (with the exception of session keys).

(iv) As observed by Rackoff (cf. [69]), a deficiency of the Bellare-Rogaway model is that the adversary is not allowed to make any queries once it has issued the 'Test' query (where it is given either a session key or a randomly selected key).

More recently, Jeong, Katz and Lee [37] proposed and analyzed a variant of the UM protocol depicted in Figure 4.2 whereby the ephemeral public keys and identities of the communicating parties are included in the key derivation function $\mathcal{H}$.



$$\kappa = \mathcal{H}(\sigma_e, \sigma_s, \hat{A}, \hat{B}, X, Y)$$

Figure 4.2: Protocol 2 — the UM variant analyzed in [37].

In [37], the BR security model was strengthened to incorporate weak forward secrecy, and to allow the adversary to issue queries even after making a Test query. Protocol 2 was proven secure in the random oracle under the CDH assumption. However, the security model and result in [37] still have the shortcomings (ii) and (iii) described above. The next section strengthens the security model to incorporate forward secrecy, resistance to malicious insider attacks, and leakage of session-specific secret information.

4.3 Security model

This section presents strengthening of the CK01 model. The definition aims to capture all essential security properties of key agreement with the exception of KCI resilience. The new definition can also be viewed as a weakening of the eCK model by the exclusion of KCI resilience. Exclusion of KCI resilience can mean that there are other security properties that are captured by the eCK definition but not in the new definition. For example, unlike the eCK definition, the new definition does not provide any assurances if the adversary learns a party's static private key and her communicating partner's ephemeral private key. However, the new definition appears to be the 'right' one for capturing all the essential security properties of the UM protocol.

The definition has been crafted specifically to allow a reductionist security proof to be given for the UM protocol. It is not expected that the definition will be useful to assess the security of other key agreement protocols. Nonetheless, the exercise of devising an appropriately strong security definition and providing a reductionist security proof for the UM protocol with respect to this definition is worthwhile given the importance of the UM protocol.

Preliminaries. In the model there are $\mathbf{n}$ parties each modeled by a probabilistic Turing machine. Each party has a static key pair together with a certificate that binds the public key to that party. The CA does not require parties to prove possession of their static private keys, but the CA verifies that the static public key of a party belongs to $\mathcal{G}^*$. Since the primary interest is to analyze the security of the UM protocol, the model is described for three-round key agreement protocols where the initiator $\hat{A}$ sends $\hat{B}$ an ephemeral public key X in the first round, $\hat{B}$ responds with an ephemeral public key Y and key confirmation tag T_B in the second round, and $\hat{A}$ sends its confirmation tag T_A in the third round. The session key is obtained by combining A, B, X, Y and possibly the identifiers $\hat{A}$, $\hat{B}$.

Notation. In the following, it is assumed that all communicated messages are represented as binary strings. The symbol $\times$ denotes a special element not in $\{0,1\}^*$. Two elements $m_1, m_2 \in \{0,1\}^* \cup \{\times\}$ are said to be *matched*, written $m_1 \sim m_2$, if either $m_1 = \times$ or $m_2 = \times$, or if $m_1 = m_2$ as binary strings. Two equal-length vectors over $\{0,1\}^* \cup \{\times\}$ are said to be matched if their corresponding components are matched.

Sessions. A party $\hat{A}$ can be activated to *create* an instance of the protocol called a *session*. A session is created via an incoming message that has one of the following forms: (i) $(\hat{A}, \hat{B})$ or (ii) $(\hat{A}, \hat{B}, Y)$. If $\hat{A}$ is activated with $(\hat{A}, \hat{B})$, then $\hat{A}$ is the session *initiator*, otherwise the session *responder*. If $\hat{A}$ is the session initiator, then $\hat{A}$ creates a separate session *state* where all session-specific short-lived information is stored, and prepares an ephemeral public key X. The session is labeled *active* and identified via a (temporary and incomplete) session *identifier* $\mathsf{s} = (\hat{A}, \hat{B}, X, \times, \times, \times)$. The outgoing message prepared by $\hat{A}$ is $(\hat{B}, \hat{A}, X)$. If $\hat{A}$ is the session responder, then $\hat{A}$ creates a separate session state and prepares an ephemeral public key X and key confirmation tag T_A. The session is labeled active and identified via a (temporary and incomplete) session identifier $\mathsf{s} = (\hat{A}, \hat{B}, Y, X, T_A, \times)$. The outgoing message is $(\hat{A}, \hat{B}, Y, X, T_A)$.

Since ephemeral keys are selected at random on a per-session basis, the probability that an ephemeral public key X is chosen twice by $\hat{A}$ is negligible. Hence session identifiers are unique except with negligible probability. For a session $(\hat{A}, \hat{B}, \mathrm{Comm}_A)$, $\hat{A}$ is the session *owner* and $\hat{B}$ is the session *peer*; together, $\hat{A}$ and $\hat{B}$ are referred to as the *communicating parties*. Here, $Comm_A$ denotes the string that comes after $\hat{A}$, $\hat{B}$ in the session identifier. The owner of a session associates a label with the session to identify whether the owner is the session's initiator or responder.

A party $\hat{A}$ can be activated to *update* an active session via an incoming message of the form (i) $(\hat{A}, \hat{B}, X, Y, T_B)$ or (ii) $(\hat{A}, \hat{B}, Y, X, T_A, T_B)$. If the message is $(\hat{A}, \hat{B}, X, Y, T_B)$, then $\hat{A}$ first checks that it owns an active session with identifier $\mathsf{s} = (\hat{A}, \hat{B}, X, \times, \times, \times)$; if not, then the message is rejected. If the session exists, then $\hat{A}$ prepares a key confirmation tag T_A, updates the identifier to $\mathsf{s} = (\hat{A}, \hat{B}, X, Y, T_B, T_A)$, and *completes* the session by accepting a session key. The outgoing message is $(\hat{B}, \hat{A}, X, Y, T_B, T_A)$. If the incoming message is $(\hat{A}, \hat{B}, Y, X, T_A, T_B)$, then $\hat{A}$ first checks that it owns an active session with identifier $\mathsf{s} = (\hat{A}, \hat{B}, Y, X, T_A, \times)$; if not, then the message is rejected. If the session exists, then $\hat{A}$ updates the identifier to $\mathsf{s} = (\hat{A}, \hat{B}, Y, X, T_A, T_B)$, and completes the session by accepting a session key. Whenever a session s completes, all information stored in its session state

is erased. Note that since the session key is not short-lived, it is not considered to be part of the session state.

Let $\mathrm{s} = (\hat{A}, \hat{B}, \mathrm{Comm}_A)$ be a session owned by $\hat{A}$. A session $\mathrm{s}^* = (\hat{C}, \hat{D}, \mathrm{Comm}_C)$ is said to be *matching* to s if $\hat{D} = \hat{A}$, $\hat{C} = \hat{B}$ and $\mathrm{Comm}_C \sim \mathrm{Comm}_A$. The session s can have more than one matching session if $\mathrm{Comm}_A = (X, \times, \times, \times)$. If $\mathrm{Comm}_A \neq (X, \times, \times, \times)$, then s can have at most one matching session (except with negligible probability) since ephemeral keys are chosen at random on a per-session basis.

A protocol may require parties to perform some checks on incoming messages. For example, if $\hat{A}$ receives the message $(\hat{A}, \hat{B}, X, Y, T_B)$, then $\hat{A}$ may need to verify that $Y \in \mathcal{G}^*$ and that T_B satisfies some authentication condition. If a party is activated to create a session with an incoming message that does not meet the protocol specifications, then that message is rejected and no session is created. If a party is activated to update an active session with an incoming message that does not meet the protocol specifications, then the party deletes all information specific to that session (including the session state and the session key if it has been computed) and *aborts* the session.

At any point of time a session is in exactly one of the following states: active, completed, aborted.

Adversary. The adversary $\mathcal{M}$ is modeled as a probabilistic Turing machine and controls *all* communications. Parties submit outgoing messages to $\mathcal{M}$, who makes decisions about their delivery. The adversary presents parties with incoming messages via *Send*(message), thereby controlling the activation of parties. The adversary does not have immediate access to a party's private information, however in order to capture possible leakage of private information $\mathcal{M}$ is allowed to make the following queries:

- *SessionStateReveal*(s): $\mathcal{M}$ obtains all the information available in the session state of s. Henceforth it is assumed that $\mathcal{M}$ issues this query only if there is some secret information in the session state of s. Since the session key is not considered to be part of the session state, $\mathcal{M}$ cannot obtain a session key with a *SessionStateReveal* query.
- *Expire*(s): If s has completed, then the session key held by s is deleted. Henceforth it is assumed that $\mathcal{M}$ issues this query only to sessions that have completed and have not yet been expired.

- *SessionKeyReveal*(s): If s has completed and has not been expired, then $\mathcal{M}$ obtains the session key held by s. Henceforth it is assumed that $\mathcal{M}$ issues this query only to sessions that have completed and have not yet been expired.
- *Corrupt*(party): $\mathcal{M}$ gains complete control over the party and is given *all* the information held by that party including its static private key, the contents of all active-session states, and all session keys (but not session keys that were deleted via an *Expire* query). In addition, $\mathcal{M}$ is able to select a new static key pair for that party. Parties against whom $\mathcal{M}$ issued a *Corrupt* query are called *corrupt* or *adversary controlled*. If a party is not corrupt, then it is said to be *honest*.

Adversary's goal. $\mathcal{M}$'s goal is to distinguish the session key held by a 'fresh' session from a random key. Informally, a session s is said to be fresh if $\mathcal{M}$ cannot determine the session key held by s by trivial means, for example by requesting it with a *SessionKeyReveal*(s) query, or by requesting it from the matching session of s should that session exists. In order to capture forward secrecy, $\mathcal{M}$ is allowed to learn the static private key of a fresh session's owner via a *Corrupt* query, but this query can be issued only after s has expired (and the session key deleted). However, to avoid capturing KCI resilience, $\mathcal{M}$ cannot obtain the static private key of a fresh session's owner *and* the ephemeral private key provided by the other communicating party (which $\mathcal{M}$ could have chosen herself, or obtained via a *SessionStateReveal* query). Formally, fresh sessions are given in the following.

Definition 4.3.1 *Let* s *be the identifier of a completed session, owned by party* $\hat{A}$ *with peer* $\hat{B}$*. Let* s^* *be the identifier of the matching session of* s*, if it exists. Define* s *to be* fresh *if none of the following conditions hold:*

1. $\mathcal{M}$ *issued* SessionKeyReveal(s)*.*
2. $\mathcal{M}$ *issued* Corrupt($\hat{A}$) *before* Expire(s)*.*
3. $\mathcal{M}$ *issued* SessionStateReveal(s) *and either* Corrupt($\hat{A}$) *or* Corrupt($\hat{B}$)*.*
4. s^* *exists and* $\mathcal{M}$ *issued one of the following:*
   (a) SessionKeyReveal(s^*)*.*
   (b) Corrupt($\hat{B}$) *before* Expire(s^*)*.*
   (c) SessionStateReveal(s^*) *and either* Corrupt($\hat{A}$) *or* Corrupt($\hat{B}$)*.*
5. s^* *does not exist and* $\mathcal{M}$ *issued* Corrupt($\hat{B}$) *before* Expire(s)*.*

As usual, the adversary $\mathcal{M}$ is allowed to make a special query *Test*(S) to a fresh session S. In response, $\mathcal{M}$ is given with equal probability either the session key held by S or a random key. $\mathcal{M}$ meets its goal if it guesses correctly whether the key is random or not. Note that $\mathcal{M}$ can continue interacting with parties after issuing the *Test* query, but must ensure that the test session remains fresh throughout $\mathcal{M}$'s experiment.

Definition 4.3.2 *A key agreement protocol is* secure *if the following conditions hold:*

1. *If two honest parties complete matching sessions, then, except with negligible probability, they both compute the same session key.*
2. *No polynomially bounded adversary $\mathcal{M}$ can distinguish the session key of a fresh session from a randomly chosen session key, with probability greater than $\frac{1}{2}$ plus a negligible fraction.*

Definition 4.3.2 overcomes the four shortcomings listed in §4.2. Although this new definition is not as strong as the eCK security definition, a reductionist security proof that a protocol satisfies Definition 4.3.2 can provide meaningful practical assurances. In particular, Definition 4.3.2 captures all elements of the CK definition, which has been accepted as a strong definition. In addition, it is stronger than the CK definition in the following ways:

1. The *SessionStateReveal* query can be issued to the test session and also to its matching session.
2. The adversary can select its own static key pair for a corrupted party, thereby allowing the modeling of malicious insider attacks.
3. The test session does not have to be unexpired at the time when the *Test* query is issued.
4. A party is allowed to execute the protocol with itself.

4.4 Protocol description

This section gives a complete description of the UM protocol, as described in SP800-56A [71]. In the following, Λ denotes optional public information that can be included in the key derivation function $\mathcal{H}$; $\mathcal{R}$ is the fixed string "KC_2_U", and $\mathcal{I}$ is the fixed string "KC_2_V"; $\hat{A}$ and $\hat{B}$ are two parties with valid public-private key pairs (A, a) and (B, b), respectively.

Definition 4.4.1 (UM protocol) *The protocol proceeds as follows:*

1. *Upon activation* $(\hat{A}, \hat{B})$*, party* $\hat{A}$ *(the initiator) performs the steps:*
   (a) *Select an ephemeral private key* $x \in_R [1, q-1]$ *and compute the ephemeral public key* $X = g^x$*.*
   (b) *Initialize the session identifier to* $(\hat{A}, \hat{B}, X, \times, \times, \times)$*.*
   (c) *Send* $(\hat{B}, \hat{A}, X)$ *to* $\hat{B}$*.*
2. *Upon activation* $(\hat{B}, \hat{A}, X)$*, party* $\hat{B}$ *(the responder) performs the steps:*
   (a) *Verify that* $X \in \mathcal{G}^*$*.*
   (b) *Select an ephemeral private key* $y \in_R [1, q-1]$ *and compute the ephemeral public key* $Y = g^y$*.*
   (c) *Compute* $\sigma_e = X^y$ *and* $\sigma_s = A^b$*. Compute* $(\kappa_m, \kappa) = \mathcal{H}(\sigma_e, \sigma_s, \hat{A}, \hat{B}, \Lambda)$*.*
   (d) *Destroy* σ_e*,* σ_s *and* y*.*
   (e) *Compute* $T_B = \mathrm{MAC}_{\kappa_m}(\mathcal{R}, \hat{B}, \hat{A}, Y, X)$*.*
   (f) *Initialize the session identifier to* $(\hat{B}, \hat{A}, X, Y, T_B)$*.*
   (g) *Send* $(\hat{A}, \hat{B}, X, Y, T_B)$ *to* $\hat{A}$*.*
3. *Upon activation* $(\hat{A}, \hat{B}, X, Y, T_B)$*,* $\hat{A}$ *performs the following steps:*
   (a) *Verify that an active session with identifier* $(\hat{A}, \hat{B}, X, \times, \times, \times)$ *exists.*
   (b) *Verify that* $Y \in \mathcal{G}^*$*.*
   (c) *Compute* $\sigma_e = Y^x$ *and* $\sigma_s = B^a$*. Compute* $(\kappa_m, \kappa) = \mathcal{H}(\sigma_e, \sigma_s, \hat{A}, \hat{B}, \Lambda)$*.*
   (d) *Destroy* σ_e*,* σ_s *and* x*.*
   (e) *Verify that* $T_B = \mathrm{MAC}_{\kappa_m}(\mathcal{R}, \hat{B}, \hat{A}, Y, X)$*.*
   (f) *Compute* $T_A = \mathrm{MAC}_{\kappa_m}(\mathcal{I}, \hat{A}, \hat{B}, X, Y)$*.*
   (g) *Destroy* κ_m*.*
   (h) *Send* $(\hat{B}, \hat{A}, X, Y, T_B, T_A)$ *to* $\hat{B}$*.*
   (i) *Update the session identifier to* $(\hat{A}, \hat{B}, X, Y, T_B, T_A)$ *and complete the session by accepting* κ *as the session key.*
4. *Upon activation* $(\hat{B}, \hat{A}, X, Y, T_B, T_A)$*,* $\hat{B}$ *performs the steps:*
   (a) *Verify that an active session with identifier* $(\hat{B}, \hat{A}, X, Y, T_B, \times)$ *exists.*
   (b) *Verify that* $T_A = \mathrm{MAC}_{\kappa_m}(\mathcal{I}, \hat{A}, \hat{B}, X, Y)$*.*

(c) *Destroy* κ_m.

(d) *Update the session identifier to* $(\hat{B}, \hat{A}, X, Y, T_B, T_A)$ *and complete the session by accepting* κ *as the session key.*

Henceforth, it is assumed that the adversary cannot issue a *SessionStateReveal*, *Expire*, *SessionKeyReveal* or *Corrupt* query while a party is executing one of the four main steps of the protocol. That is, the adversary can only issue one of these queries at the end of Steps 1, 2, 3 or 4. This means that a *SessionStateReveal* query can yield x (at the end of Step 1) or κ_m (at the end of Step 2), but not y. In order to account for possible loss of y to the adversary via a side-channel attack or the use of a weak pseudorandom number generator, henceforth the adversary can learn y by issuing a *SessionStateReveal* query at the end of Step 2 even though Step 2 stipulates that y be deleted.

4.5 Security argument

For simplicity, the case $\Lambda = (X, Y)$, where X and Y are the exchanged ephemeral public keys, will be considered first. Furthermore, the security argument assumes that a party does not initiate a session with itself. These restrictions will be relaxed in §4.5.1 and §4.5.2.

Theorem 4.5.1 *Suppose that* $\mathcal{G}$ *is a group where the GDH assumption holds, that the* MAC *scheme is secure, and that* $\mathcal{H}$ *is modeled as a random oracle. Then the UM protocol (Definition 4.4.1) is secure in the sense of Definition 4.3.2.*

Proof: Verifying condition 1 of Definition 4.3.2 is straightforward; it remains to verify condition 2 of Definition 4.3.2. Let λ denote the security parameter, and let $\mathcal{M}$ be a polynomially (in λ) bounded UM adversary. Assume that $\mathcal{M}$ succeeds in an environment with $\mathbf{n}$ parties, activates a party to create a session at most $\mathbf{s}$ times, and terminates after time $\mathcal{T}_{\mathcal{M}}$. Here, $\mathbf{n}$ and $\mathbf{s}$ are bounded by polynomials in λ. Let M denote the event that $\mathcal{M}$ succeeds, and suppose that $\Pr(M) = \frac{1}{2} + p(\lambda)$ where $p(\lambda)$ is non-negligible. Following the standard approach such an adversary $\mathcal{M}$ is used to construct a polynomial-time algorithm $\mathcal{S}$ that, with non-negligible probability of success, solves a CDH instance (U, V) or breaks the MAC scheme.

Since $\mathcal{H}$ is a random function, $\mathcal{M}$ has two possible strategies for winning its distinguishing game with probability significantly greater than $\frac{1}{2}$:

(i) induce a non-matching session s' to establish the same session key as the test session s, and thereafter issue a *SessionKeyReveal*(s') query; or

(ii) query the random oracle $\mathcal{H}$ with $(g^{xy}, g^{ab}, \hat{A}, \hat{B}, X, Y)$ where $\mathrm{s} = (\hat{A}, \hat{B}, X, Y, T_B, T_A)$ is the test session or its matching session.

Now, two sessions $(\hat{A}, \hat{B}, X, Y, T_B, T_A)$ and $(\hat{B}, \hat{A}, X, Y, T'_B, T'_A)$ cannot both be initiators or responders except with negligible probability. It follows that $T_B = T'_B$ and $T_A = T'_A$, and so the sessions are matching. Hence, since the input to the key derivation function includes the identities of the communicating parties and the exchanged ephemeral public keys, non-matching completed sessions produce different session keys except with negligible probability of $\mathcal{H}$ collisions. This rules out strategy (i). Now, let H^* denote the event that $\mathcal{M}$ queries the random oracle $\mathcal{H}$ with $(g^{xy}, g^{ab}, \hat{A}, \hat{B}, X, Y)$ where $\mathrm{s} = (\hat{A}, \hat{B}, X, Y, T_B, T_A)$ is the test session or its matching session. Since $\mathcal{H}$ is a random function, it follows that

$$\Pr(M|\overline{H^*}) = \frac{1}{2}$$

where negligible terms are ignored. Hence

$$\begin{aligned} \Pr(M) &= \Pr(M \wedge H^*) + \Pr(M|\overline{H^*}) \Pr(\overline{H^*}) \\ &\leq \Pr(M \wedge H^*) + \frac{1}{2}, \end{aligned}$$

whence $\Pr(M \wedge H^*) \geq p(\lambda)$. We will henceforth denote the event $M \wedge H^*$ by M^*.

Let s^t denote the test session selected by $\mathcal{M}$, and let s^m denote its matching session (if it exists). Consider the following complementary events:

E_1. s^m exists, and $\mathcal{M}$ issues neither *SessionStateReveal*(s^t) nor *SessionStateReveal*(s^m).

E_2. Either s^m does not exist, or $\mathcal{M}$ issues *SessionStateReveal*(s^t), or $\mathcal{M}$ issues *SessionStateReveal*(s^m).

Since $\Pr(M^*)$ is non-negligible, it must be the case that either $p_1(\lambda) = \Pr(M^* \wedge E_1)$ or $p_2(\lambda) = \Pr(M^* \wedge E_2)$ is non-negligible. The events E_1 and E_2 are considered separately.

The following conventions will be used in the remainder of this section: $\xi : \mathcal{G} \times \mathcal{G} \rightarrow \mathcal{G}$ is a random function known only to $\mathcal{S}$ and such that $\xi(X, Y) = \xi(Y, X)$ for all $X, Y \in \mathcal{G}$. The algorithm $\mathcal{S}$, which simulates $\mathcal{M}$'s environment, will use $\xi(U, Z)$ to represent $\mathrm{CDH}(U, Z)$ in situations where $\mathcal{S}$ does not know $\log_g U$. Except with negligible probability, $\mathcal{M}$ will not detect that $\xi(U, Z)$ is being used instead of $\mathrm{CDH}(U, Z)$.

Simulation in event E_1. In this scenario, $\mathcal{S}$ establishes $\mathbf{n}$ parties, who are assigned random static key pairs, and selects $s_1, s_2 \in_R [1, \ldots, \mathbf{ns}]$. The s_1'th and s_2'th sessions created will be called s^U and s^V, respectively. The adversary $\mathcal{M}$ is activated on this set of $\mathbf{n}$ parties and the simulation of $\mathcal{M}$'s environment proceeds as follows.

1. *Send*$(\hat{A}, \hat{B})$: $\mathcal{S}$ executes Step 1 of the protocol. However, if the session being created is the s_1'th or s_2'th session, then $\mathcal{S}$ deviates from the protocol description by setting the ephemeral public key X to be U or V, respectively; note that $\mathcal{S}$ does not possess the corresponding ephemeral private key in this case.

2. *Send*$(\hat{B}, \hat{A}, X)$: $\mathcal{S}$ executes Step 2 of the protocol. However, if the session being created is the s_1'th or s_2'th session, then $\mathcal{S}$ deviates from the protocol description by setting the ephemeral public key Y to be U or V, respectively, and setting $\sigma_e = \xi(Y, X)$; note that $\mathcal{S}$ does not possess the corresponding ephemeral private key in this case.

3. *Send*$(\hat{A}, \hat{B}, X, Y, T_B)$: $\mathcal{S}$ executes Step 3 of the protocol. However, if $X \in \{U, V\}$, then $\mathcal{S}$ deviates from the protocol description by setting $\sigma_e = \xi(X, Y)$.

4. *Send*$(\hat{B}, \hat{A}, X, Y, T_B, T_A)$: $\mathcal{S}$ executes Step 4 of the protocol.

5. *SessionStateReveal*(s): $\mathcal{S}$ answers the query faithfully except if $\mathrm{s} \in \{\mathrm{s}^U, \mathrm{s}^V\}$ in which case $\mathcal{S}$ aborts with failure.

6. *Expire*(s): $\mathcal{S}$ answers the query faithfully.

7. *SessionKeyReveal*(s): $\mathcal{S}$ answers the query faithfully except if $\mathrm{s} \in \{\mathrm{s}^U, \mathrm{s}^V\}$ in which case $\mathcal{S}$ aborts with failure.

8. *Corrupt*$(\hat{A})$: If $\hat{A}$ owns session s^U or s^V, and that session is not expired, then $\mathcal{S}$ aborts with failure. Otherwise, $\mathcal{S}$ answers the query faithfully.

9. $\mathcal{H}(\sigma_e, \sigma_s, \hat{A}, \hat{B}, X, Y)$:

   (a) If $X \in \{U, V\}$ and $\sigma_e \neq \xi(X, Y)$, then $\mathcal{S}$ obtains $\tau = \mathrm{DDH}(X, Y, \sigma_e)$.

      i. If $\tau = 0$, then $\mathcal{S}$ simulates a random oracle in the usual way.

      ii. If $\tau = 1$ and $Y \in \{U, V\}$ and $Y \neq X$, then $\mathcal{S}$ aborts with success and outputs $\mathrm{CDH}(U, V) = \sigma_e$.

      iii. Otherwise, if either $Y \notin \{U, V\}$ or $Y = X$, then $\mathcal{S}$ returns $\mathcal{H}(\xi(X, Y), \sigma_s, \hat{A}, \hat{B}, X, Y)$.

   (b) If $Y \in \{U, V\}$ and $\sigma_e \neq \xi(X, Y)$, then $\mathcal{S}$ obtains $\tau = \mathrm{DDH}(X, Y, \sigma_e)$.

      i. If $\tau = 0$, then $\mathcal{S}$ simulates a random oracle in the usual way.

ii. If $\tau = 1$, then $\mathcal{S}$ returns $\mathcal{H}(\xi(X,Y), \sigma_s, \hat{A}, \hat{B}, X, Y)$.

(c) $\mathcal{S}$ simulates a random oracle in the usual way.

10. *Test*(s): If $\mathrm{s} \notin \{\mathrm{s}^U, \mathrm{s}^V\}$ or if s^U and s^V are non-matching, then $\mathcal{S}$ aborts with failure. Otherwise, $\mathcal{S}$ answers the query faithfully.

Analysis of event $E_1 \wedge M^*$. $\mathcal{S}$'s simulation of $\mathcal{M}$'s environment is perfect except with negligible probability. The probability that $\mathcal{M}$ selects one of s^U, s^V as the test session and the other as its matching session is least $2/(\mathbf{ns})^2$. Suppose that this is indeed the case, and suppose that event $M^* \wedge E_1$ occurs. Then $\mathcal{S}$ does not abort as described in Steps 5 and 10. Furthermore, since the test session is fresh, $\mathcal{S}$ does not abort as described in Steps 7 and 8.

Except with negligible probability of guessing $\xi(U,V)$, a successful $\mathcal{M}$ must query $\mathcal{H}$ with $(\mathrm{CDH}(U,V), \mathrm{CDH}(X,Y), \hat{A}, \hat{B}, X, Y)$ where $\{X,Y\} = \{U,V\}$, in which case $\mathcal{S}$ is successful as described in Step 9(a). The probability that $\mathcal{S}$ is successful is bounded by

$$\mathcal{P}(\mathcal{S}) \geq \frac{2}{(\mathbf{ns})^2} p_1(\lambda), \tag{4.1}$$

where negligible terms are ignored.

Event E_2. Let F be the event "s^m does not exist and $\mathcal{M}$ does not issue *SessionStateReveal*(s^t)". Event E_2 is further subdivided into the following complementary events:

E_{2a}. $E_2 \wedge \overline{F}$.

E_{2b}. $E_2 \wedge F$.

Let $p_{2a}(\lambda) = \mathcal{P}(M^* \wedge E_{2a})$ and $p_{2b}(\lambda) = \mathcal{P}(M^* \wedge E_{2b})$, whence $p_2 = p_{2a} + p_{2b}$. If event $M^* \wedge E_2$ occurs with non-negligible probability, then either $M^* \wedge E_{2a}$ or $M^* \wedge E_{2b}$ occurs with non-negligible probability. The events E_{2a} and E_{2b} are considered separately. In both cases, $\mathcal{S}$ establishes $\mathbf{n}$ parties. Two of these parties, denoted $\hat{U}$ and $\hat{V}$, are randomly selected and assigned static public keys U and V, respectively. Note that $\mathcal{S}$ does not know the corresponding static private keys. The other $\mathbf{n} - 2$ parties are assigned random static key pairs.

Simulation in event E_{2a}. The adversary $\mathcal{M}$ is activated on the set of $\mathbf{n}$ parties and the simulation of $\mathcal{M}$'s environment proceeds as follows.

1. *Send*$(\hat{A}, \hat{B})$: $\mathcal{S}$ executes Step 1 of the protocol.
2. *Send*$(\hat{B}, \hat{A}, X)$: $\mathcal{S}$ executes Step 2 of the protocol. However, if $\hat{B} \in \{\hat{U}, \hat{V}\}$, then $\mathcal{S}$ deviates from the protocol description by setting $\sigma = \xi(X, Y)$.
3. *Send*$(\hat{A}, \hat{B}, X, Y, T_B)$: $\mathcal{S}$ executes Step 3 of the protocol. However, if $\hat{A} \in \{\hat{U}, \hat{V}\}$, then $\mathcal{S}$ deviates from the protocol description by setting $\sigma_s = \xi(X, Y)$.
4. *Send*$(\hat{B}, \hat{A}, X, Y, T_B, T_A)$: $\mathcal{S}$ executes Step 4 of the protocol.
5. *SessionStateReveal*(S): $\mathcal{S}$ answers the query faithfully.
6. *Expire*(S): $\mathcal{S}$ answers the query faithfully.
7. *SessionKeyReveal*(S): $\mathcal{S}$ answers the query faithfully.
8. *Corrupt*$(\hat{A})$: If $\hat{A} \in \{\hat{U}, \hat{V}\}$, then $\mathcal{S}$ aborts with failure. Otherwise, $\mathcal{S}$ answers the query faithfully.
9. $\mathcal{H}(\sigma_e, \sigma_s, \hat{A}, \hat{B}, X, Y)$:
   a. If $\hat{A} \in \{\hat{U}, \hat{V}\}$ and $\sigma_s \neq \xi(X, Y)$, then $\mathcal{S}$ obtains $\tau = \mathrm{DDH}(A, B, \sigma_s)$.
      i. If $\tau = 0$, then $\mathcal{S}$ simulates a random oracle in the usual way.
      ii. If $\tau = 1$ and $\hat{B} \in \{\hat{U}, \hat{V}\}$ and $\hat{B} \neq \hat{A}$, then $\mathcal{S}$ aborts with success and outputs $\mathrm{CDH}(U, V) = \sigma_s$.
      iii. Otherwise, if $\hat{B} \notin \{\hat{U}, \hat{V}\}$ or $\hat{B} = \hat{A}$, then $\mathcal{S}$ returns $\mathcal{H}(\sigma_e, \xi(A, B), \hat{A}, \hat{B}, X, Y)$.
   (b) If $\hat{B} \in \{\hat{U}, \hat{V}\}$ and $\sigma_s \neq \xi(A, B)$, then $\mathcal{S}$ obtains $\tau = \mathrm{DDH}(A, B, \sigma_s)$.
      i. If $\tau = 0$, then $\mathcal{S}$ simulates a random oracle in the usual way.
      ii. If $\tau = 1$, then $\mathcal{S}$ returns $\mathcal{H}(\sigma_e, \xi(A, B), \hat{A}, \hat{B}, X, Y)$.
   (c) $\mathcal{S}$ simulates a random oracle in the usual way.
10. *Test*(S): If the communicating parties of S are not $\hat{U}$ and $\hat{V}$, then $\mathcal{S}$ aborts with failure. Otherwise, $\mathcal{S}$ answers the query faithfully.

Analysis of event $E_{2a} \wedge M^*$. $\mathcal{S}$'s simulation of $\mathcal{M}$'s environment is perfect except with negligible probability. The probability that $\hat{U}$ and $\hat{V}$ are the communicating parties of the test session selected by $\mathcal{M}$ is at least $2/\mathbf{n}^2$. Suppose that this is indeed the case, so $\mathcal{S}$ does not abort in Step 10, and suppose that event $M^* \wedge E_{2a}$ occurs. Now, if $\mathcal{M}$ issued a *SessionStateReveal*(s^t) query, then, by definition of a fresh session, $\mathcal{M}$ cannot have corrupted $\hat{U}$ or $\hat{V}$. On the other hand, if $\mathcal{M}$ did not issue a *SessionStateReveal*(s^t) query, then, by definition of event $\overline{F}$, s^m must exist. Consequently, by definition of event E_2, $\mathcal{M}$ must have issued a *SessionStateReveal*(s^m) query, and hence cannot have corrupted $\hat{U}$ or $\hat{V}$. Hence, $\mathcal{S}$ does not abort as described in Step 8.

Except with negligible probability of guessing $\xi(U, V)$, a successful $\mathcal{M}$ must query $\mathcal{H}$ with $(\mathrm{CDH}(X, Y), \mathrm{CDH}(U, V), \hat{A}, \hat{B}, X, Y)$ where $\{\hat{A}, \hat{B}\} = \{\hat{U}, \hat{V}\}$, in which case $\mathcal{S}$ is successful as described in Step 9(a). The probability that $\mathcal{S}$ is successful is therefore bounded by

$$\mathcal{P}(\mathcal{S}) \geq \frac{2}{\mathbf{n}^2} p_{2a}(\lambda), \tag{4.2}$$

where negligible terms are ignored.

Simulation in event E_{2b}. $\mathcal{S}$ is given a MAC oracle with key $\tilde{\kappa_m}$ that is unknown to $\mathcal{S}$. $\mathcal{S}$ selects $r \in_R [1, \ldots, \mathbf{ns}]$ and activates $\mathcal{M}$. The rth session created will be called s^r. The adversary $\mathcal{M}$ is activated on this set of parties and the simulation of $\mathcal{M}$'s environment proceeds as follows.

1. *Send*($\hat{A}, \hat{B}$): $\mathcal{S}$ executes Step 1 of the protocol. If the session created is the rth session and $\{\hat{A}, \hat{B}\} \neq \{\hat{U}, \hat{V}\}$, then $\mathcal{S}$ aborts with failure.

2. *Send*($\hat{B}, \hat{A}, X$): $\mathcal{S}$ executes Step 2 of the protocol. However, if $\hat{B} \in \{\hat{U}, \hat{V}\}$, then $\mathcal{S}$ deviates from the protocol description by setting $\sigma_s = \xi(A, B)$.

   If the created session is the rth session, then $\mathcal{S}$ deviates from the protocol description as follows. If $\{\hat{A}, \hat{B}\} \neq \{\hat{U}, \hat{V}\}$, then $\mathcal{S}$ aborts with failure. Otherwise, $\mathcal{S}$ selects a random session key κ and sets the MAC key κ_m equal to the (unknown) key $\tilde{\kappa_m}$ of the MAC oracle. $\mathcal{S}$ queries the MAC oracle with $(\mathcal{R}, \hat{B}, \hat{A}, Y, X)$ and sets T_B equal to the oracle response.

3. *Send*($\hat{A}, \hat{B}, X, Y, T_B$): $\mathcal{S}$ executes Step 3 of the protocol. However, if $\hat{A} \in \{\hat{U}, \hat{V}\}$, then $\mathcal{S}$ deviates from the protocol description by setting $\sigma_s = \xi(A, B)$. If $\hat{A}$ was activated to update s^r, then $\mathcal{S}$ selects a random session key κ and sets the MAC key κ_m equal to the (unknown) key $\tilde{\kappa_m}$ of the MAC oracle. $\mathcal{S}$

queries the MAC oracle with $(\mathcal{I}, \hat{A}, \hat{B}, X, Y)$ and sets T_A equal to the oracle response.

4. *Send*$(\hat{B}, \hat{A}, X, Y, T_B, T_A)$: $\mathcal{S}$ executes Step 4 of the protocol. If $\hat{B}$ was activated to update s^r, then $\mathcal{S}$ completes the session without verifying the received T_A.

5. *SessionStateReveal*(s): $\mathcal{S}$ answers the query faithfully. However, if $\mathrm{s} = \mathrm{s}^r$ and the owner of s^r is the session responder, then $\mathcal{S}$ aborts with failure.

6. *Expire*(s): $\mathcal{S}$ answers the query faithfully. However, if $\mathrm{s} = \mathrm{s}^r$ and s^r does not have a matching session, then $\mathcal{S}$ aborts with success and outputs as its MAC forgery the key confirmation tag received by s (and the associated message). If $\mathrm{s} = \mathrm{s}^r$ and s^r has a matching session, then $\mathcal{S}$ aborts with failure.

7. *SessionKeyReveal*(s): $\mathcal{S}$ answers the query faithfully.

8. *Corrupt*$(\hat{A})$: If $\hat{A} \in \{\hat{U}, \hat{V}\}$, then $\mathcal{S}$ aborts with failure. Otherwise, $\mathcal{S}$ answers the query faithfully.

9. $\mathcal{H}(\sigma_e, \sigma_s, \hat{A}, \hat{B}, X, Y)$:

   (a) If $\hat{A} \in \{\hat{U}, \hat{V}\}$ and $\sigma_s \neq \xi(A, B)$, then $\mathcal{S}$ obtains $\tau = \mathrm{DDH}(A, B, \sigma_s)$.

      i. If $\tau = 0$, then $\mathcal{S}$ simulates a random oracle in the usual way.

      ii. If $\tau = 1$ and $\hat{B} \in \{\hat{U}, \hat{V}\}$ and $\hat{B} \neq \hat{A}$, then $\mathcal{S}$ aborts with success and outputs $\mathrm{CDH}(U, V) = \sigma_s$.

      iii. Otherwise, if $\hat{B} \notin \{\hat{U}, \hat{V}\}$ or $\hat{B} = \hat{A}$, then $\mathcal{S}$ returns $\mathcal{H}(\sigma_e, \xi(A, B), \hat{A}, \hat{B}, X, Y)$.

   (b) If $\hat{B} \in \{\hat{U}, \hat{V}\}$ and $\sigma_s \neq \xi(A, B)$, then $\mathcal{S}$ obtains $\tau = \mathrm{DDH}(A, B, \sigma_s)$.

      i. If $\tau = 0$, then $\mathcal{S}$ simulates a random oracle in the usual way.

      ii. If $\tau = 1$, then $\mathcal{S}$ returns $\mathcal{H}(\sigma_e, \xi(A, B), \hat{A}, \hat{B}, X, Y)$.

   (c) $\mathcal{S}$ simulates a random oracle in the usual way.

10. *Test*(s): If $\mathrm{s} \neq \mathrm{s}^r$ or if s^r has a matching session, then $\mathcal{S}$ aborts with failure. Otherwise, $\mathcal{S}$ answers the query faithfully.

Analysis of Event $E_{2b} \wedge M^*$. $\mathcal{S}$'s simulation of $\mathcal{M}$'s environment is perfect except with negligible probability. The probability that the test session is the rth session, and $\hat{U}$ and $\hat{V}$ are its communicating parties, is at least $2/(\mathbf{n}^3\mathbf{s})$. Suppose that this is indeed the case so $\mathcal{S}$ does not abort in Steps 1 and 2, and suppose that event $M^* \wedge E_{2b}$ occurs so $\mathcal{S}$ does not abort in Step 6. By definition of event F, $\mathcal{S}$ does not

abort in Steps 5 and 10. Also by definition of a fresh session, $\mathcal{M}$ is only allowed to corrupt either $\hat{U}$ or $\hat{V}$ after expiring the test session. Therefore before aborting as in Step 8, $\mathcal{S}$ is successful as in Step 6.

Except with negligible probability of guessing $\xi(U, V)$, a successful $\mathcal{M}$ must query $\mathcal{H}$ with $(\mathrm{CDH}(X, Y), \mathrm{CDH}(U, V), \hat{A}, \hat{B}, X, Y)$ where $\{\hat{A}, \hat{B}\} = \{\hat{U}, \hat{V}\}$, in which case $\mathcal{S}$ is successful as described in Step 9(a). The probability that $\mathcal{S}$ is successful is bounded by

$$\mathcal{P}(\mathcal{S}) \geq \frac{2}{\mathbf{n}^3\mathbf{s}} p_{2b}(\lambda), \tag{4.3}$$

where negligible terms are ignored.

Overall analysis. Suppose event M occurs. Combining Equations 4.1, 4.2 and 4.3, for every adversary $\mathcal{M}$ there is an algorithm $\mathcal{S}$ that solves the GDH problem or breaks the MAC scheme with success probability $\mathcal{P}(\mathcal{S})$, where

$$\mathcal{P}(\mathcal{S}) \geq \max\left\{ \frac{2}{(\mathbf{ns})^2} p_1(\lambda),\ \frac{2}{\mathbf{n}^2} p_{2a}(\lambda),\ \frac{2}{\mathbf{n}^3\mathbf{s}} p_{2b}(\lambda) \right\}. \tag{4.4}$$

During the simulations, $\mathcal{S}$ performs group exponentiations and MAC computations, accesses the DDH oracle, and simulates a random oracle. Let $q = \Theta(2^\lambda)$. Then a group exponentiation takes time $\mathcal{T}_\mathcal{G} = O(\lambda)$ group multiplications. Assume that a MAC computation, a DDH oracle call, and a response to an $\mathcal{H}$ query take polynomial time, $\mathcal{T}_{\mathrm{MAC}}(\lambda)$, $\mathcal{T}_{\mathrm{DDH}}(\lambda)$, and $\mathcal{T}_\mathcal{H}(\lambda)$, respectively. The running time $\mathcal{T}_\mathcal{S}$ of $\mathcal{S}$ is therefore bounded by

$$\mathcal{T}_\mathcal{S} \leq (2\mathcal{T}_\mathcal{G} + 2\mathcal{T}_{\mathrm{MAC}} + \mathcal{T}_{\mathrm{DDH}} + \mathcal{T}_\mathcal{H})\, \mathcal{T}_\mathcal{M}. \tag{4.5}$$

Thus, if $\mathcal{M}$ is polynomially bounded, then there is an algorithm $\mathcal{S}$ that succeeds in solving the GDH problem in $\mathcal{G}$ or breaks the MAC scheme with non-negligible probability. Furthermore, $\mathcal{S}$ runs in polynomial time contradicting the assumptions of Theorem 4.5.1. This concludes the argument. □

4.5.1 Reflections

In the simulations of E_{2a} and E_{2b} it was implicitly assumed (in Step 9(a)) that $\hat{U}$ and $\hat{V}$ are distinct parties. More precisely, if a party is allowed to initiate a session with itself, then $\mathcal{S}$ may fail as $\mathcal{M}$ may produce $\mathrm{CDH}(U, U)$ or $\mathrm{CDH}(V, V)$ instead of $\mathrm{CDH}(U, V)$. The case $\hat{U} = \hat{V}$ can be encompassed by a reduction from the Gap Square Problem (GSP). $\mathcal{S}$'s actions are modified as follows. Given $U = g^u$, $\mathcal{S}$ selects $v \in_R [1 \ldots q-1]$ and computes $V = U^v$. The output produced by $\mathcal{S}$ in event E_1 is

$\sigma_e^{v^{-1}}$. In events E_{2a} and E_{2b}, $\mathcal{S}$'s output is $\sigma_s^{v^{-1}}$ if the communicating parties are $\hat{U}$ and $\hat{V}$, σ_s if $\hat{U}$ is both the owner and peer of the test session, and $\sigma_s^{v^{-2}}$ if $\hat{V}$ is both the owner and peer of the test session.

4.5.2 No ephemeral public keys in the KDF

If ephemeral public keys are not included in the key derivation function (i.e., if Λ is the empty string), then the following attack on the UM protocol can be launched by $\mathcal{M}$.

1. $\mathcal{M}$ induces two sessions $\mathrm{s}^1 = (\hat{A}, \hat{B}, X, Y, T_B, T_A)$ and $\mathrm{s}^2 = (\hat{B}, \hat{A}, X, Y, T_B, T_A)$ to complete, where $\hat{A}$ is the initiator and $\hat{B}$ is the responder. Note that $(\kappa_m, \kappa) = \mathcal{H}(g^{xy}, g^{ab}, \hat{A}, \hat{B})$. During the protocol run, $\mathcal{M}$ learns the ephemeral private keys x, y and the MAC key κ_m via *SessionStateReveal* queries to s^1 and s^2.

2. $\mathcal{M}$ issues *Test*(s^1).

3. $\mathcal{M}$ issues *Send*($\hat{A}, \hat{B}$). In response, $\hat{A}$ selects ephemeral key pair (x^*, X^*) where $X^* = g^{x^*}$, initiates session $\mathrm{s} = (\hat{A}, \hat{B}, X^*, \times, \times, \times)$, and sends $(\hat{B}, \hat{A}, X^*)$.

4. $\mathcal{M}$ issues *SessionStateReveal*(s) to learn x^*, and computes $y^* = xy(x^*)^{-1}$, $Y^* = g^{y^*}$, and the MAC tag $T_B^* = \mathrm{MAC}_{\kappa_m}(\mathcal{R}, \hat{B}, \hat{A}, Y^*, X^*)$.

5. $\mathcal{M}$ issues *Send*($\hat{A}, \hat{B}, X^*, Y^*, T_B^*$). Since $g^{x^*y^*} = g^{xy}$, $\hat{A}$ computes the same (κ_m, κ) pair as she computed for session s^1. Thus the tag T_B^* is valid, and $\hat{A}$ completes session s with session key κ.

6. $\mathcal{M}$ now obtains κ by issuing *SessionKeyReveal*(s), and thereafter correctly answers the *Test* query (note that session s^1 is still fresh).

The attack relies on the ability of the adversary to obtain (temporary) MAC keys κ_m via a *SessionStateReveal* query. If MAC keys are deemed to be at risk, then the ephemeral public keys X and Y should be included in the key derivation function, thus thwarting attacks like the one described above. Such attacks can also be prevented if the responder $\hat{B}$ computes T_A and deletes κ_m in Step 2 of the protocol, and then uses the stored copy of T_A to verify $\hat{A}$'s tag in Step 4. In this way the attacker does not learn κ_m via a *SessionStateReveal* query. Suppose now that the adversary is unable to obtain MAC keys. The security proof can be modified

for the case where the ephemeral public keys are not included in the key derivation function. A short description of the required changes follows.

A potential problem is that the adversary $\mathcal{M}$ could force two non-matching sessions to compute the same session key, issue the *Test* query to one session, and learn the session key from the other session. Now, the test session completes only after obtaining the correct key confirmation tag. Since this tag contains the identifiers of the communicating parties, the exchanged ephemeral public keys, and the string $\mathcal{R}$ or $\mathcal{I}$ identifying whether the tag was created by an initiator or responder, $\mathcal{M}$ cannot fool the test session owner into completing a session by reusing a MAC tag from a non-matching session. Since the MAC algorithm is assumed to be secure, it must be the case that $\mathcal{M}$ computed the MAC key itself by querying $\mathcal{H}$ with $(\mathrm{CDH}(U,V), \mathrm{CDH}(A,B), \hat{A}, \hat{B})$ in case E_1 and with $(\mathrm{CDH}(X,Y), \mathrm{CDH}(U,V), \hat{A}, \hat{B})$ in cases E_{2a} and E_{2b}. To complete the proof the simulation in event E_1 has to be modified as follows: whenever $\mathcal{M}$ queries $\mathcal{H}$ with $(\sigma_e, \sigma_s, \hat{A}, \hat{B})$, $\mathcal{S}$ checks whether $\hat{A}$ or $\hat{B}$ owns either of s^U or s^V. If so, then $\mathcal{S}$ uses the DDH oracle to test if $\mathrm{CDH}(U,V) = \sigma_e$, in which case $\mathcal{S}$ is successful.

4.6 Concluding remarks

This chapter provided a reductionist security argument for the UM protocol with respect to a strengthened version of the CK01 model. The reduction is not tight, but that is perhaps unavoidable given that there can be many parties and sessions. Furthermore, given a desired security level, it is not clear how to use the reduction in Theorem 4.5.1 to derive concrete recommendations for the parameters of the cryptographic ingredients.

Chapter 5

Combined model

5.1 Motivation

In the CK01 model, the BR model, the BJM model and the eCK models, and associated definitions, key agreement protocols are analyzed in the so-called *pre-specified peer model* wherein it is assumed that a party knows the identifier of its intended communicating peer when it commences a run of the protocol. That is, it is assumed that the exchange of identifiers, and possibly also the static public keys of the communicating parties, is handled by the application that invokes a run of the protocol.

On the other hand, a party in the post-specified peer model for key agreement does not know the identifier of its communicating peer at the outset, but learns the identifier during the protocol run. In 2002, Canetti and Krawczyk [21] introduced the *post-specified peer model* (also abbreviated CK02) wherein a party is activated to establish a session key knowing only a destination address (such as the IP address of a server) of the communicating peer, and only learns the peer's identifier during the execution of the protocol. According to [21], this scenario is common in practical settings where the peer's identifier is simply unavailable at the outset, or if one party wishes to conceal its identity from eavesdroppers or active adversaries. The IKE protocols [31, 27] (see also [40]) are important examples of key agreement protocols that provide the option of identity concealment.

In the remainder of this chapter the identity concealment attribute of key agreement protocols will not be considered. Furthermore, 'pre-specified peer model' will be shortened to 'pre model', and 'post-specified peer model' will be shortened to 'post model'.

A key agreement protocol designed for one of the pre or post models is *executable* in the other model if it can be run in the second model without requiring any additional message flows (and without making any fundamental changes to the protocol description). Any key agreement protocol designed for the post model is executable in the pre model. Indeed, if the peer's identifier and static public key is not needed at the start of the protocol, then the protocol can also be executed given the peer's identifier. Canetti and Krawczyk observed that the Σ_0 key agreement protocol is secure in the post model with respect to the security definition given in [21], but not secure in the pre model with respect to the security definition given in [20]. Hence, even though any protocol designed for the post model can be executed in the pre model, security in the post model of [21] does not guarantee security in the pre model of [20]; further details are given in §5.3.3.

This chapter explores the executability and security in the post model of key agreement protocols that have been designed for and analyzed in the pre model. Of course any protocol designed for the pre model can be modified for the post model by adding message flows which include the identifiers and static public keys of the communicating parties; however such a modification does not conform to the "executable" notion here because of the additional message flows. An example of a key agreement protocol that is secure in the pre model but is not executable in the post model is given. Furthermore, the HMQV protocol [41], which has been proven to be secure in the pre model, is executable in the post model without additional message flows but is not secure. These examples illustrate the essential differences between the two models, and highlight the danger of running in the post model a protocol that has only been analyzed in the pre model.

It is natural then to ask when a protocol secure in one model is executable and secure in the other model. §5.4.1 identifies a class of *modifiable* key agreement protocols that have been designed for the pre model but can be executed with minimal modifications in the post model. This class includes many of the protocols that have been proposed in the literature including station-to-station [25], UM [71], MQV [47], Boyd-Mao-Paterson [15], HMQV [41], KEA+ [46], NAXOS [45], CMQV [73] and Okamoto [59]. (See [16] for an extensive list of key establishment protocols.) Such protocols have a *hybrid* description that combine the specification for the pre model and the specification of the modified protocol suitable for the post model. A *combined* model and associated security definition developed in this chapter aim to simultaneously capture the security assurances (and more) of the extended Canetti-Krawczyk pre-specified peer model [45] and the Canetti-Krawczyk post-specified peer model [21]. The combined model has the feature that if a hybrid key agreement protocol is proven secure in that model, then its specializations are

guaranteed to be secure when run in the pre and post models.

5.2 The CK02 model

The Canetti-Krawczyk pre-specified peer model (CK01) is outlined in §2.2.1. The Canetti-Krawczyk post-specified peer model (CK02) and associated security definition [21] are essentially the same as in the CK01 model, but there are two important differences.

First, a session at $\hat{A}$ is created via a message containing (at least) three parameters $(\hat{A}, \tilde{d}, \Psi)$, where $\tilde{d}$ is a *destination address* to which outgoing messages should be delivered. That is, party $\hat{A}$ does not know the identifier of its peer when it starts the session. During the course of the protocol run, $\hat{A}$ learns the (alleged) identifier $\hat{B}$ of the communicating party; this party is referred to as $\hat{A}$'s peer for that session.

Second, the definition of a matching session is different. Let $(\hat{A}, \Psi)$ be a session that has completed with peer $\hat{B}$. Then a session $(\hat{B}, \Psi)$ is said to be *matching* to $(\hat{A}, \Psi)$ if either (i) $(\hat{B}, \Psi)$ has not yet completed; or (ii) $(\hat{B}, \Psi)$ has completed and its peer is $\hat{A}$. Condition (i) is necessary because the incomplete session $(\hat{B}, \Psi)$ may not yet have determined its peer and hence could have been communicating with $(\hat{A}, \Psi)$, in which case exposure of $(\hat{B}, \Psi)$ could possibly reveal non-trivial information about the session key held by $(\hat{A}, \Psi)$.

5.3 CK01 and CK02 differences

This section presents three examples to illustrate the differences between the Canetti-Krawczyk security definitions for key agreement in the pre- and post- specified peer models.

5.3.1 A CK01 to CK02 non-adaptable example

As argued above, any protocol designed for the CK02 model can be trivially adapted for the CK01 model. However, the converse is not true.

Recall the μ-protocol from §2.3, which according to Theorem 2.3.1 is secure in the pre model. Observe that in the μ-protocol the initiator $\hat{A}$ cannot prepare the first outgoing message without knowledge of the peer's identifier $\hat{B}$ and static public key B. Hence, unless protocol μ is modified in a fundamental way, it cannot be executed in the post-specified peer model without additional message flows to exchange identifiers and static public keys.

5.3.2 A CK01 to CK02 adaptable example

HMQV [41], which was informally described in §3.3, is an efficient two-pass Diffie-Hellman key agreement protocol that has been proven to be secure in the pre-specified peer model under the CDH and KEA1 assumptions and where the hash functions employed are modeled as random functions. The security definition used in [41] is stronger than the security definition outlined in §2.2.1 in the sense that the adversary is granted certain additional capabilities. For example, the adversary is allowed to register a static key pair at any time thus allowing the modeling of attacks by malicious insiders.

Unlike protocol μ, HMQV can be readily adapted to run in the post-specified peer model. Indeed, the initiator can prepare the first message (which essentially consists of the ephemeral public key X) without knowledge of the peer's identifier $\hat{B}$ or static public key Y. It is natural then to ask whether HMQV is secure in the post model. This is also important because the version of HMQV that is being considered for standardization by the P1363 working group [42] does not mandate that the protocol be executed in the pre model, in other words there is no requirement that the communicating parties possess each other's identifiers and static public keys prior to a run of the protocol. Consequently, HMQV may in fact be executed in the post model in applications where the responder's identifier is not available to the initiator at the beginning of the protocol run.

The attack described next demonstrates that HMQV, without further modifications such as the addition of message flows to exchange identifiers and static public keys, is not secure in the post model. The attack makes the following plausible assumptions: (i) the group order q is a 160-bit prime and so the outputs of $\mathcal{H}_2$ have bitlength 80; (ii) the best attack on the CDH problem in $\mathcal{G}$ takes approximately 2^{80} steps; (iii) there are at least 2^{20} honest (uncorrupted) parties; (iv) a party can select its own identifier; and (v) the certification authority does not require parties to prove knowledge of the static private keys corresponding to their static public keys during registration. In [41] it is noted that the HMQV security proof does not depend on the CA performing any proof-of-possession checks. The attack proceeds as follows.

1. The adversary $\mathcal{M}$ induces $\hat{A}$ to create a session with a destination address $\hat{d}$ controlled by $\mathcal{M}$. In response, $\hat{A}$ selects ephemeral key pair (x, X) and sends $(\hat{d}, \hat{A}, X)$.

2. $\mathcal{M}$ intercepts $(\hat{d}, \hat{A}, X)$ and does the following:

(a) Compute $S = \{(\hat{C}, \mathcal{H}_2(X, \hat{C})) \mid \hat{C} \text{ is an honest party}\}$.

(b) Select an identifier $\hat{M}$ (not the same as the identifier of an honest party) such that $(\hat{B}, \mathcal{H}_2(X, \hat{M})) \in S$ for some $\hat{B}$.

(c) Select $M = B$ as $\hat{M}$'s static public key (note that $\mathcal{M}$ does not know the corresponding private key).

(d) Send $(\hat{B}, \hat{A}, X)$ to $\hat{B}$.

3. $\mathcal{M}$ intercepts $\hat{B}$'s reply $(\hat{A}, \hat{B}, Y)$ and sends $(\hat{A}, \hat{M}, Y)$ to $\hat{A}$.

Party $\hat{A}$ computes the session key $\kappa = \mathcal{H}(\sigma_A)$, where $\sigma_A = (YM^{\mathrm{E}})^{x+\mathrm{E}a}$ and $\mathrm{E} = \mathcal{H}_2(X, \hat{M})$ and $\mathrm{D} = \mathcal{H}_2(Y, \hat{A})$. Party $\hat{B}$ computes the session key $\kappa' = \mathcal{H}(\sigma_B)$, where $\sigma_B = (XA^{\mathrm{D}'})^{y+\mathrm{E}'b}$ and $\mathrm{E}' = \mathcal{H}_2(X, \hat{B})$ and $\mathrm{D}' = \mathcal{H}_2(Y, \hat{A})$. Since $\mathrm{E}' = \mathrm{E}$, $\mathrm{D}' = \mathrm{D}$, and $M = B$, it follows that $\sigma_A = \sigma_B$ and hence $\kappa' = \kappa$. The problem is that while $\hat{B}$ correctly believes that κ is shared with $\hat{A}$, party $\hat{A}$ mistakenly believes that κ is shared with $\hat{M}$. Thus $\mathcal{M}$ has successfully launched a UKS attack on HMQV in the post model. The expected running time of the attack is about 2^{60} (for step 2b). Since most of the work has to be done on-line, the attack cannot be considered practical. Nevertheless it demonstrates that HMQV does not attain an 80-bit security level in the post model as it presumably does in the pre model.

The mechanisms of the attack were outlined in Remark 7.2 of [41]. However, the adversary considered in [41] operates in a different setting, namely the pre model where party $\hat{A}$ precomputes and stores her ephemeral public keys X which are then inadvertently leaked to $\mathcal{M}$ *before* $\hat{A}$ uses them in a session. Three countermeasures were proposed in [41] for foiling this attack: (i) increase the output length of $\mathcal{H}_2$ to 160 bits; (ii) include the identifiers $\hat{A}$, $\hat{B}$ in the key derivation function whereby the session key is computed as $\kappa = \mathcal{H}(\sigma, \hat{A}, \hat{B})$; and (iii) include random nonces (which are not precomputed and stored) in the derivation of exponents E and D, whereby the exponents are computed as $\mathrm{E} = \mathcal{H}_2(X, \hat{B}, \nu_A)$ and $\mathrm{D} = \mathcal{H}(Y, \hat{A}, \nu_B)$ where ν_A and ν_B are $\hat{A}$'s and $\hat{B}$'s nonces, respectively. Countermeasures (i) and (ii) are successful in thwarting the attack described above on HMQV in the post model. However, it can easily be seen that countermeasure (iii) does *not* prevent the attack in the post model, thus demonstrating that the two attacks are indeed different. The reason countermeasure (iii) fails is that, unlike in the pre model, the peer's identifier is not known to $\hat{A}$ when $\hat{A}$ creates the session in the post model.

5.3.3 A CK02 to CK01 example

The Σ_0 protocol [21] is a simplified version of one of the IKE key agreement protocols. In the protocol description below, PRF is a pseudorandom function family,

MAC is a message authentication code algorithm, and sig_A and sig_B are the signing algorithms for $\hat{A}$ and $\hat{B}$, respectively.

1. Party $\hat{A}$ (the initiator) selects an ephemeral key pair (x, X), initializes the session identifier to $(\hat{A}, \Psi)$, and sends $(\hat{d}_B, \hat{d}_A, \Psi, X)$. Here $\hat{d}_A$ and $\hat{d}_B$ are destination addresses for $\hat{A}$ and $\hat{B}$, respectively.

2. Upon receipt of $(\hat{d}_B, \hat{d}_A, \Psi, X)$, $\hat{B}$ (the responder) selects an ephemeral key pair (y, Y), and computes $\sigma_e = X^b$, $\kappa = \mathrm{PRF}_{\sigma_e}(0)$, and $\kappa_m = \mathrm{PRF}_{\sigma_e}(1)$. $\hat{B}$ then destroys y and σ_e, initializes the session identifier to $(\hat{B}, \Psi)$, and sends $m_1 = (\hat{d}_A, \hat{B}, \Psi, Y, \mathrm{sig}_B(\mathcal{R}, \Psi, X, Y), \mathrm{MAC}_{\kappa_m}(\mathcal{R}, \Psi, \hat{B}))$.

3. Upon receiving m_1, $\hat{A}$ computes $\sigma_e = Y^a$, $\kappa = \mathrm{PRF}_{\sigma_e}(0)$, and $\kappa_m = \mathrm{PRF}_{\sigma_e}(1)$. $\hat{A}$ then verifies the signature and MAC tag in m_1, and sends m_2, where

$$m_2 = (\hat{B}, \hat{A}, \Psi, \mathrm{sig}_A(\mathcal{I}, \Psi, Y, X), \mathrm{MAC}_{\kappa_m}(\mathcal{I}, \Psi, \hat{A})).$$

   Finally, $\hat{A}$ accepts the session key κ with peer $\hat{B}$, and erases the session state.

4. Upon receiving m_2, $\hat{B}$ verifies the signature and MAC tag in m_2, accepts the session κ with peer $\hat{A}$, and erases the session state.

In [21], the Σ_0 protocol is proven secure in the post-specified peer model provided that the DDH assumption holds in $\mathcal{G}$ and that the PRF, MAC, and sig primitives are secure. However, the following attack described in [21] shows that Σ_0 is not secure in the pre-specified peer model.

1. Create a session $(\hat{A}, \hat{B}, \Psi)$ at $\hat{A}$.

2. Intercept $\hat{A}$'s outgoing message $(\hat{B}, \hat{A}, \Psi, X)$ and send $(\hat{B}, \hat{M}, \Psi, X)$ to $\hat{B}$.

3. Intercept $\hat{B}$'s response $(\hat{M}, \hat{B}, \Psi, Y, S_B, T_B)$, where $S_B = \mathrm{sig}_B(\mathcal{R}, \Psi, X, Y)$ and $T_B = \mathrm{MAC}_{\kappa_m}(\mathcal{R}, \Psi, \hat{B})$, and send $(\hat{A}, \hat{B}, \Psi, Y, S_B, T_B)$ to $\hat{A}$.

4. The session $(\hat{A}, \hat{B}, \Psi)$ at $\hat{A}$ completes and accepts κ as the session key.

5. Intercept and delete $\hat{A}$'s final message, and issue a *SessionStateReveal* query to the session $(\hat{B}, \hat{M}, \Psi)$ thus learning κ and κ_m.

6. Issue the *Test* query to the session $(\hat{A}, \hat{B}, \Psi)$ and use knowledge of κ to win the distinguishing game.

Notice that the attack is legitimate in the pre-specified peer model since the exposed session $(\hat{B}, \hat{M}, \Psi)$ is not matching to the test session $(\hat{A}, \hat{B}, \Psi)$. On the other hand, such an attack is not permitted in the post-specified peer model because in Step 5 of the attack the session $(\hat{B}, \Psi)$ is still incomplete and therefore matching to the *Test* session (and thus cannot be exposed). This is all rather counterintuitive since one would expect that if a protocol is secure when the initiator does not have a priori knowledge of the peer's identifier, then it should remain secure when the peer's identifier is known at the outset.

One feature of both the pre and post models is that an exposed session does not produce any further output. In practice, however, one might desire the assurance that a particular session is secure even if the adversary learns some secret state information (such as an ephemeral private key) associated with that session or its matching session. For this reason, the security models in recent papers such as [41], [45] and [73] permit exposed sessions to continue producing output, and furthermore allow the adversary to issue a *SessionStateReveal* query (or its equivalent) to the *Test* session and its matching session (cf. §5.4.3). However, if the adversary $\mathcal{M}$ is equipped with these extra capabilities, then the Σ_0 protocol would be insecure in both the pre and the post models since $\mathcal{M}$ could issue *SessionStateReveal* query to $(\hat{A}, \Psi)$ after Step 1 to learn x and thereafter compute the session key. The attack is a little more realistic than the attack described above in the pre model because the new attack assumes that the *SessionStateReveal* query does not yield the session key κ (which may be stored in secure memory). $\mathcal{M}$'s actions are the following:

1. Create a session $(\hat{A}, \Psi)$ at $\hat{A}$ with peer destination address $\hat{d}_B$.
2. Intercept $\hat{A}$'s outgoing message $(\hat{d}_B, \hat{d}_A, \Psi, X)$ and send $(\hat{d}_B, \hat{d}_M, \Psi, X)$ to $\hat{B}$.
3. Intercept $\hat{B}$'s response $(\hat{d}_M, \hat{B}, \Psi, Y, S_B, T_B)$, where $S_B = \mathrm{sig}_B(\mathcal{R}, \Psi, X, Y)$ and $T_B = \mathrm{MAC}_{\kappa_m}(\mathcal{R}, \Psi, \hat{B})$, and send $(\hat{d}_A, \hat{B}, \Psi, Y, S_B, T_B)$ to $\hat{A}$.
4. Intercept $\hat{A}$'s final message and delete it. The session $(\hat{A}, \Psi)$ completes with peer $\hat{B}$ and session key κ.
5. Issue a *SessionStateReveal* query to the incomplete session $(\hat{B}, \Psi)$ and learn the MAC key κ_m.
6. Compute $S_M = \mathrm{sig}_M(\mathcal{I}, \Psi, Y, X)$ and $T_M = \mathrm{MAC}_{\kappa_m}(\mathcal{I}, \Psi, \hat{M})$, and send $(\hat{B}, \hat{M}, \Psi, S_M, T_M)$ to $\hat{B}$.

The session $(\hat{B}, \Psi)$ completes with peer $\hat{M}$ and session key κ. Thus $\mathcal{M}$ has successfully launched a UKS attack on Σ_0 in the post model. The last attack demon-

strates that a protocol proven secure in the post-specified peer model of [21] may no longer be secure if exposed sessions are allowed to continue producing output.

5.4 Combining and extending CK01 and CK02

This section introduces the notion of a modifiable key agreement protocol — protocols, with two or more flows, designed for the pre-specified peer model but which can be adapted with minor changes to be executable in the post-specified peer model. This leads to the notion of a hybrid key agreement protocol, which simultaneously describes a modifiable protocol and its modification suitable for the post model. Following that the section develops a security definition that, if satisfied by a hybrid protocol, guarantees that the associated protocols are secure in the pre and post models.

5.4.1 Modifiable protocols

Consider a key agreement protocol Π designed for the pre model where the first outgoing message prepared by the initiator $\hat{A}$ is of the form $(\hat{B}, \hat{A}, \mathit{RoundOne})$. Then Π is said to be *modifiable* if *RoundOne* can be computed before the session is created at $\hat{A}$; in particular, this means that *RoundOne* does not depend on $\hat{B}$'s identifier or static public key.

A modifiable protocol Π can be easily adapted for the post-specified peer model by incorporating identity establishment into the protocol flows. The required changes are the following. The initiator $\hat{A}$, who is activated to create a session with a destination address $\hat{d}$ (and without knowledge of the recipient's identifier or static public key), sends $(\hat{d}, \hat{A}, \mathit{RoundOne})$ as $\hat{A}$'s first outgoing message. Since this message contains the identifier $\hat{A}$, the responder has all the information he needs to prepare his first outgoing message as specified by Π. The responder appends his identifier to this outgoing message (if the message does not already contain the identifier). After $\hat{A}$ receives this reply, both $\hat{A}$ and the responder can proceed with Π without any further modifications. Notice that the modified protocol Π' has the same number of message flows as the original protocol Π.

As mentioned in §5.1, the class of modifiable key agreement protocols includes many of the protocols that have been proposed in the literature. However, not all key agreement protocols are modifiable; for example, protocol μ defined in §2.3 is not modifiable. Furthermore, as demonstrated by the attack on HMQV in §5.3.2,

security of a modifiable protocol Π in the pre model does not imply security of the modified protocol Π' in the post model.

5.4.2 Hybrid protocols

Suppose that Π is a modifiable key agreement protocol, and Π' its modification suitable for the post model. The specification of Π and Π' can be combined as described below, resulting in a protocol $\tilde{\Pi}$ called a *hybrid* protocol.

Let $\tilde{A}$ denote either an identifier $\hat{A}$ or a destination address $\hat{d}$ that can be used to send messages to some party $\hat{A}$ whose identifier is not known to the sender; note that the address $\hat{d}$ may not necessarily be under $\hat{A}$'s control. In the description of $\tilde{\Pi}$, a session is created at initiator $\hat{A}$ via a message containing $(\hat{A}, \tilde{B})$. The first outgoing message from $\hat{A}$ is $(\tilde{B}, \hat{A}, \mathit{RoundOne})$. The responder $\hat{B}$ includes the identifiers $\hat{A}$ and $\hat{B}$ in his response, and the remainder of the protocol description is the same as for Π.

A hybrid protocol $\tilde{\Pi}$ can be specialized for the pre model by using an identifier $\hat{B}$ for $\tilde{B}$. Protocol $\tilde{\Pi}$ can also be specialized for the post model by using a destination address for $\tilde{B}$. An example of a hybrid protocol is given in §5.5.

5.4.3 Combined security model

The combined model and associated security definition aim to simultaneously capture the security assurances of the pre- and post-specified peer models. That is, if a hybrid protocol $\tilde{\Pi}$ is proven secure with respect to the new definition, then its specializations Π and Π' are guaranteed to be secure when run in the pre and post models, respectively. More precisely, the eCK model can be obtained by restricting the queries in the combined model, and hence, when run in the pre model, Π satisfies the eCK definition suitably enhanced to capture attacks where an adversary is able to learn ephemeral public keys of parties *before* they are actually used in a protocol session. As discussed in [41], such attacks may be possible in situations where a party precomputes ephemeral public keys in order to improve on-line performance. Such attacks were considered by Krawczyk [41], but were not incorporated into his security model.

When run in the post model, the modified protocol Π' satisfies a strengthened version of the CK02 definition, suitably enhanced to offer security assurances similar to the eCK definition (including resistance to attacks where the adversary learns

ephemeral private keys of the session being attacked) and to capture attacks where the adversary learns ephemeral public keys before they are actually used.

Instead of using pre-determined session numbers Ψ to identify sessions (cf. §2.2.1), the session identifiers will consist of the identities of the communicating parties together with a concatenation of the messages exchanged during a protocol run. As shown in [23], this notion of session identifier yields a security model for key agreement that is at least as strong as other security models.

Notation and terminology. Assume that messages are represented as binary strings. If m is a vector, then $\#m$ denotes the number of its components. Two vectors m_1 and m_2 are *matched*, written $m_1 \sim m_2$, if the first $t = \min\{\#m_1, \#m_2\}$ components of the vectors are pairwise equal as binary strings. Furthermore, $\hat{A} \equiv \tilde{D}$ if either $\tilde{D} = \hat{A}$ or if $\tilde{D}$ is a destination address that can be used to send messages to $\hat{A}$.

Session creation. A party $\hat{A}$ can be activated via an incoming message to create a session. The incoming message has one of the following forms: (i) $(\hat{A}, \tilde{B})$ or (ii) $(\tilde{A}, \hat{B}, \mathit{In})$. If $\hat{A}$ was activated with $(\hat{A}, \tilde{B})$, then $\hat{A}$ is the session initiator; otherwise $\hat{A}$ is the session responder.

Session initiator. If $\hat{A}$ is the session initiator, then $\hat{A}$ creates a separate session state where session-specific short-lived data is stored, and prepares a reply *Out* that includes an ephemeral public key X. The session is labeled active and identified via a (temporary and incomplete) session identifier $\mathsf{s} = (\hat{A}, \tilde{B}, \mathcal{I}, \mathrm{Comm})$ where Comm is initialized to *Out*. The outgoing message is $(\tilde{B}, \hat{A}, \mathit{Out})$.

Session responder. If $\hat{A}$ is the session responder, then $\hat{A}$ creates a separate session state and prepares a reply *Out* that includes an ephemeral public key X. The session is labeled active and identified via a (temporary and incomplete) session identifier $\mathsf{s} = (\hat{A}, \hat{B}, \mathcal{R}, \mathrm{Comm})$ where $\mathrm{Comm} = (\mathit{In}, \mathit{Out})$. The outgoing message is $(\hat{B}, \hat{A}, \mathcal{I}, \mathit{In}, \mathit{Out})$.

Session update. A party $\hat{A}$ can be activated to update a session via an incoming message of the form $(\hat{A}, \hat{B}, role, \mathrm{Comm}, \mathit{In})$, where $role \in \{\mathcal{I}, \mathcal{R}\}$. Upon receipt of this message, $\hat{A}$ checks that she owns an active session with identifier $\mathsf{s} = (\hat{A}, \hat{B}, role, \mathrm{Comm})$ or $\mathsf{s} = (\hat{A}, \hat{d}, role, \mathrm{Comm})$ where $\hat{d}$ is a destination address;

except with negligible probability, $\hat{A}$ can own at most one such session. If no such session exists, then the message is rejected. If a session $\mathrm{s} = (\hat{A}, \hat{d}, role, \mathrm{Comm})$ or $\mathrm{s} = (\hat{A}, \hat{B}, role, \mathrm{Comm})$ exists, then in the former case $\hat{A}$ updates the session identifier to $\mathrm{s} = (\hat{A}, \hat{B}, role, \mathrm{Comm})$; in either case, $\hat{A}$ updates s by appending *In* to Comm. If the protocol requires a response by $\hat{A}$, then $\hat{A}$ prepares the required response *Out*; the outgoing message is $(\hat{B}, \hat{A}, role, \mathrm{Comm}, \mathit{Out})$ where $role$ is $\hat{B}$'s role as perceived by $\hat{A}$, and the session identifier is updated by appending *Out* to Comm. If the protocol specifies that no further messages will be received, then the session completes and accepts a session key.

Matching sessions. Since ephemeral public keys are selected at random on a per-session basis, session identifiers are unique except with negligible probability. Party $\hat{A}$ is said to be the owner of a session $(\hat{A}, \tilde{B}, *, *)$. For a session $(\hat{A}, \hat{B}, *, *)$ we call $\hat{B}$ the session *peer*; together $\hat{A}$ and $\hat{B}$ are referred to as the *communicating parties*. Let $\mathrm{s} = (\hat{A}, \tilde{B}, role_A, \mathrm{Comm}_A)$ be a session owned by $\hat{A}$, where $role_A \in \{\mathcal{I}, \mathcal{R}\}$. A session $\mathrm{s}^* = (\hat{C}, \tilde{D}, role_C, \mathrm{Comm}_C)$, where $role_C \in \{\mathcal{I}, \mathcal{R}\}$, is said to be *matching* to s if $\hat{C} \equiv \tilde{B}$, $\hat{A} \equiv \tilde{D}$, $role_A \neq role_C$, and $\mathrm{Comm}_C \sim \mathrm{Comm}_A$. It can be seen that the session s, except with negligible probability, can have more than one matching session if and only if Comm_A has exactly one component, i.e., is comprised of a single outgoing message.

Aborted sessions. A protocol may require parties to perform some checks on incoming messages. For example, a party may be required to perform some form of public key validation or verify a signature. If a party is activated to create a session with an incoming message that does not meet the protocol specifications, then that message is rejected and no session is created. If a party is activated to update an active session with an incoming message that does not meet the protocol specifications, then the party deletes all information specific to that session (including the session state and the session key if it has been computed) and *aborts* the session; such an abortion occurs before the session identifier can be updated. At any point in time a session is in exactly one of the following states: active, completed, aborted.

Adversary. The adversary $\mathcal{M}$ is modeled as a probabilistic Turing machine and controls *all* communications. In particular, this means that $\hat{A} \equiv \hat{d}$ for all parties $\hat{A}$ and all destination addresses $\hat{d}$. Parties submit outgoing messages to $\mathcal{M}$, who makes decisions about their delivery. The adversary presents parties with incoming messages via *Send*(message), thereby controlling the activation of parties. The

adversary does not have immediate access to a party's private information, however in order to capture possible leakage of private information $\mathcal{M}$ is allowed to make the following queries:

- *StaticKeyReveal*($\hat{A}$): $\mathcal{M}$ obtains $\hat{A}$'s static private key.
- *EphemeralKeyReveal*(S): $\mathcal{M}$ obtains the ephemeral private key held by session S. Henceforth, it is assumed that $\mathcal{M}$ issues this query only to sessions that hold an ephemeral private key.
- *SessionKeyReveal*(S): If S has completed, then $\mathcal{M}$ obtains the session key held by S. Henceforth, it is assumed that $\mathcal{M}$ issues this query only to sessions that have completed.
- *EphemeralPublicKeyReveal*($\hat{A}$): $\mathcal{M}$ obtains the ephemeral public key that $\hat{A}$ will use the next time a session is created within $\hat{A}$.
- *Establish*($\hat{M}$,M): This query allows $\mathcal{M}$ to register an identifier $\hat{M}$ and a static public key M on behalf of a party. The adversary totally controls that party, thus permitting the modeling of attacks by malicious insiders. Parties that were established by $\mathcal{M}$ using *Establish* are called *corrupted* or *adversary controlled*. If a party is not corrupted it is said to be *honest*.

Adversary's goal. To capture the indistinguishability requirement, $\mathcal{M}$ is allowed to make a special query *Test*(S) to a 'fresh' session S. In response, $\mathcal{M}$ is given with equal probability either the session key held by S or a random key. $\mathcal{M}$ meets its goal if it guesses correctly whether the key is random or not. Note that $\mathcal{M}$ can continue interacting with the parties after issuing the *Test* query, but must ensure that the test session remains fresh throughout $\mathcal{M}$'s experiment.

Definition 5.4.1 *Let* S *be the identifier of a completed session, owned by an honest party* $\hat{A}$ *with peer* $\hat{B}$*, who is also honest. Let* S* *be the identifier of the matching session of* S*, if it exists. Define* S *to be* fresh *if none of the following conditions hold:*

1. $\mathcal{M}$ *issued* SessionKeyReveal(S) *or* SessionKeyReveal(S*) *(if* S* *exists).*
2. S* *exists and* $\mathcal{M}$ *issued one of the following:*
   (a) *Both* StaticKeyReveal($\hat{A}$) *and* EphemeralKeyReveal(S)*.*
   (b) *Both* StaticKeyReveal($\hat{B}$) *and* EphemeralKeyReveal(S*)*.*

3. S^* *does not exist and* $\mathcal{M}$ *issued one of the following:*

   (a) *Both* StaticKeyReveal($\hat{A}$) *and* EphemeralKeyReveal(S)*.*

   (b) StaticKeyReveal($\hat{B}$)*.*

Definition 5.4.2 *A key agreement protocol is* secure *if the following conditions hold:*

1. *If two honest parties complete matching sessions, then, except with negligible probability, they both compute the same session key.*
2. *No polynomially bounded adversary* $\mathcal{M}$ *can distinguish the session key of a fresh session from a randomly chosen session key, with probability greater than* $\frac{1}{2}$ *plus a negligible fraction.*

5.5 A hybrid example

The hybrid version of the NAXOS-C key agreement protocol is essentially the NAXOS protocol (see [45]) augmented with key confirmation. NAXOS-C can be specialized to run in either the pre or the post model. Moreover, it is secure in the combined model of §5.4.3 provided that the GDH assumption holds in $\mathcal{G}$ and that the hash functions $\mathcal{H}$, $\mathcal{H}_1$ and $\mathcal{H}_2$ are modeled as random functions. Hence NAXOS-C is secure in both the pre- and post-specified peer models.

The purpose of presenting the NAXOS-C protocol is to demonstrate that the security definition of §5.4.3 is useful (and not too restrictive) in the sense that there exist practical protocols that meet the definition under reasonable assumptions. The protocol was designed to allow a straightforward (albeit tedious) reductionist security argument, and has not been optimized. In particular, not all the inputs to the hash functions $\mathcal{H}$, $\mathcal{H}_1$ and $\mathcal{H}_2$ may be necessary for security, and in practice $\mathcal{H}_2$ would be implemented as a MAC algorithm (with secret key κ_m).

5.5.1 NAXOS-C description

In the protocol description, λ is the security parameters, and $\mathcal{H} : \{0,1\}^* \to \{0,1\}^\lambda \times \{0,1\}^\lambda$, $\mathcal{H}_1 : \{0,1\}^* \to [1, q-1]$, and $\mathcal{H}_2 : \{0,1\}^* \to \{0,1\}^\lambda$ are hash functions.

Definition 5.5.1 (NAXOS-C protocol) *The protocol proceeds as follows:*

1. *Upon activation* $(\hat{A}, \tilde{B})$*, party* $\hat{A}$ *(the initiator) performs the steps:*

(a) *Select an ephemeral private key* $\tilde{x} \in_R \{0,1\}^\lambda$*, and compute* $x = \mathcal{H}_1(a, \tilde{x})$ *and* $X = g^x$.

(b) *Destroy* x.

(c) *Initialize the session identifier to* $(\hat{A}, \tilde{B}, \mathcal{I}, X)$.

(d) *Send* $(\tilde{B}, \hat{A}, X)$ *to* $\tilde{B}$.

2. *Upon activation* $(\tilde{B}, \hat{A}, X)$*, party* $\hat{B}$ *(the responder) performs the steps:*

(a) *Verify that* $X \in \mathcal{G}^*$.

(b) *Select an ephemeral private key* $\tilde{y} \in_R \{0,1\}^\lambda$*, and compute* $y = \mathcal{H}_1(b, \tilde{y})$ *and* $Y = g^b$.

(c) *Compute* $\sigma_1 = A^y$*,* $\sigma_2 = X^b$ *and* $\sigma_e = X^y$.

(d) *Compute* $(\kappa, \kappa_m) = \mathcal{H}(\hat{A}, \hat{B}, A, B, \sigma_1, \sigma_2, \sigma_e)$ *and* $T_B = \mathcal{H}_2(\kappa_m, \mathcal{R}, \hat{B}, \hat{A}, Y, X)$.

(e) *Destroy* $\tilde{y}, y, \sigma_1, \sigma_2$ *and* σ_e.

(f) *Initialize the session identifier to* $(\hat{B}, \hat{A}, \mathcal{R}, X, Y, T_B)$.

(g) *Send* $(\hat{A}, \hat{B}, \mathcal{I}, X, Y, T_B)$ *to* $\hat{A}$.

3. *Upon activation* $(\hat{A}, \hat{B}, \mathcal{I}, X, Y, T_B)$*,* $\hat{A}$ *performs the steps:*

(a) *Verify that an active session with identifier* $(\hat{A}, \tilde{B}, \mathcal{I}, X)$ *exists.*

(b) *Verify that* $Y \in \mathcal{G}^*$.

(c) *Compute* $x = \mathcal{H}_1(a, \tilde{x})$*,* $\sigma_1 = Y^a$*,* $\sigma_2 = B^x$ *and* $\sigma_e = Y^x$.

(d) *Compute* $(\kappa, \kappa_m) = \mathcal{H}(\hat{A}, \hat{B}, A, B, \sigma_1, \sigma_2, \sigma_e)$.

(e) *Destroy* $\tilde{x}, x, \sigma_1, \sigma_2$ *and* σ_e.

(f) *Verify that* $T_B = \mathcal{H}_2(\kappa_m, \mathcal{R}, \hat{B}, \hat{A}, Y, X)$.

(g) *Compute* $T_A = \mathcal{H}_2(\kappa_m, \mathcal{I}, \hat{A}, \hat{B}, X, Y)$.

(h) *Destroy* κ_m.

(i) *Send* $(\hat{B}, \hat{A}, \mathcal{R}, X, Y, T_B, T_A)$ *to* $\hat{B}$.

(j) *Update the session identifier to* $(\hat{A}, \hat{B}, \mathcal{I}, X, Y, T_B, T_A)$ *and complete the session by accepting* κ *as the session key.*

4. *Upon activation* $(\hat{B}, \hat{A}, \mathcal{R}, X, Y, T_B, T_A)$*,* $\hat{B}$ *performs the steps:*

(a) *Verify that an active session with identifier* $(\hat{B}, \hat{A}, \mathcal{R}, X, Y, T_B)$ *exists.*

(b) *Verify that* $T_A = \mathcal{H}_2(\kappa_m, \mathcal{I}, \hat{A}, \hat{B}, X, Y)$.

(c) *Destroy* κ_m.

(d) *Update the session identifier to* $(\hat{B}, \hat{A}, \mathcal{R}, X, Y, T_B, T_A)$ *and complete the session by accepting* κ *as the session key.*

If any of the verification steps fail the party erases all session specific information and aborts the session.

Henceforth, it is assumed that the adversary cannot issue a *StaticKeyReveal*, *EphemeralKeyReveal*, *SessionKeyReveal*, *EphemeralPublicKeyReveal* or *Establish* query while a party is executing one of the four main steps of the protocol. That is, the adversary can only issue one of these queries at the end of Steps 1, 2, 3 or 4.

5.5.2 Security arguments for NAXOS-C

The security of NAXOS-C is established in the following theorem:

Theorem 5.5.1 *If* $\mathcal{H}$, $\mathcal{H}_1$ *and* $\mathcal{H}_2$ *are modeled as random oracles, and* $\mathcal{G}$ *is a group where the* GDH *assumption holds, then NAXOS-C is secure in the combined model.*

Proof: Verifying that NAXOS-C satisfies condition 1 of Definition 5.4.2 is straightforward. It remains to verify that condition 2 of Definition 5.4.2 is satisfied – that no polynomially bounded adversary can distinguish the session key of a fresh session from a randomly chosen session key. Let λ denote the security parameter, and let $\mathcal{M}$ be a polynomially (in λ) bounded adversary. The adversary $\mathcal{M}$ is said to be successful with non-negligible probability if $\mathcal{M}$ wins the distinguishing game with probability $\frac{1}{2} + p(\lambda)$, where $p(\lambda)$ is non-negligible. The event that $\mathcal{M}$ is successful is denoted by M. Following the standard approach such an adversary $\mathcal{M}$ will be used to construct an algorithm $\mathcal{S}$ that with non-negligible probability of success solves a CDH instance (U, V) in $\mathcal{G}$.

Let the test session be $\mathrm{s} = (\hat{A}, \hat{B}, \mathcal{I}, X, Y, T_B, T_A)$ or $\mathrm{s} = (\hat{B}, \hat{A}, \mathcal{R}, X, Y, T_B, T_A)$, and let H^* be the event that $\mathcal{M}$ queries $\mathcal{H}$ with $(\hat{A}, \hat{B}, X, Y, \sigma_1, \sigma_2, \sigma_e)$. Let $\overline{H^*}$ be the complement of event H^*, and let s^* be any completed session owned by an honest party such that $\mathrm{s}^* \neq \mathrm{s}$ and s^* is non-matching to s. Since s^* and s are distinct and non-matching, it can be seen that the inputs to the key derivation function $\mathcal{H}$ are different for s and s^*. And, since $\mathcal{H}$ is a random oracle, it follows that $\mathcal{M}$

cannot obtain any information about the test session key from the session keys of non-matching sessions. Hence $\Pr(M \wedge \overline{H^*}) \leq \frac{1}{2}$ and

$$\Pr(M) = \Pr(M \wedge H^*) + \Pr(M \wedge \overline{H^*}) \leq \Pr(M \wedge H^*) + \frac{1}{2},$$

whence $\Pr(M \wedge H^*) \geq p(\lambda)$. Henceforth, event $M \wedge H^*$ will be denoted by M^*.

Assume further that $\mathcal{M}$ succeeds in an environment with $\mathbf{n}$ parties, activates at most $\mathbf{s}$ sessions within a party, makes at most h, h_1, h_2 queries to oracles $\mathcal{H}$, $\mathcal{H}_1$ and $\mathcal{H}_2$ respectively, and terminates after time at most $\mathcal{T}_{\mathcal{M}}(\lambda)$.

The following conventions will be used for the remainder of the security argument: $\xi : \mathcal{G} \times \mathcal{G} \rightarrow \mathcal{G}$ is a random function known only to $\mathcal{S}$, such that $\xi(X, Y) = \xi(Y, X)$ for all $X, Y \in \mathcal{G}$. The algorithm $\mathcal{S}$, which simulates $\mathcal{M}$'s environment, will use $\xi(X, Y)$ to "represent" $\mathrm{CDH}(X, Y)$ in situations where $\mathcal{S}$ may not know $\log_g X$ or $\log_g Y$. Except with negligible probability, $\mathcal{M}$ will not detect that $\xi(X, Y)$ is being used instead of $\mathrm{CDH}(X, Y)$.

Consider the following complementary events:

E_1: There exists an honest party $\hat{B}$ such that $\mathcal{M}$, during its execution, queries $\mathcal{H}_1$ with $(b, *)$ before issuing a *StaticKeyReveal*($\hat{B}$) query. (Note that $\mathcal{M}$ does not necessarily make a *StaticKeyReveal*($\hat{B}$) query.)

E_2: During its execution, for every party $\hat{B}$ for which $\mathcal{M}$ queries $\mathcal{H}_1$ with $(b, *)$, $\mathcal{M}$ issued *StaticKeyReveal*($\hat{B}$) before the first $(b, *)$ query to $\mathcal{H}_1$.

Since $\Pr(M^*)$ is non-negligible, it must be the case that either event $E_1 \wedge M^*$ or $E_2 \wedge M^*$ occurs with non-negligible probability. These possibilities are considered separately.

Simulation in event $E_1 \wedge M^*$. Suppose that event $E_1 \wedge M^*$ occurs with non-negligible probability. $\mathcal{S}$ begins by establishing $\mathbf{n}$ parties. One of these parties, denoted $\hat{V}$, is selected at random and assigned the static public key V, and $\nu \in_R [1, q-1]$ is used to represent the corresponding static private key. The remaining parties are assigned random static key pairs. For each honest party $\hat{A}$, $\mathcal{S}$ maintains a list of at most $\mathbf{s}$ ephemeral key pairs, and two markers – a party marker and an adversary marker. The list is initially empty, and the markers initially point to the first entry of the list. Whenever $\hat{A}$ is activated to create a new session, $\mathcal{S}$ checks if the party marker points to an empty entry. If so then $\mathcal{S}$ selects a new ephemeral key pair on behalf of $\hat{A}$ as described in Step 1 or 2 of the NAXOS-C protocol. If

the list entry is not empty, then $\mathcal{S}$ uses the ephemeral key pair in that list entry for the newly created session. In either case the party marker is updated to point to the next list entry, and the adversary marker is also advanced if it points to an earlier entry. If $\mathcal{M}$ issues an *EphemeralPublicKeyReveal* query, then $\mathcal{S}$ selects a new ephemeral key pair on behalf of $\hat{A}$ as described in Step 1 or 2 of the NAXOS-C protocol. $\mathcal{S}$ stores the key pair in the entry pointed to by the adversary marker, returns the public key as the query response, and advances the adversary marker.

With this setup $\mathcal{S}$ activates the adversary $\mathcal{M}$ on this set of parties and awaits the actions of $\mathcal{M}$. The simulation of $\mathcal{M}$'s environment proceeds as follows.

1. *Send*$(\hat{A}, \tilde{B})$: $\mathcal{S}$ executes Step 1 of the protocol.
2. *Send*$(\tilde{B}, \hat{A}, X)$ issued to $\hat{B}$: $\mathcal{S}$ executes Step 2 of the protocol. However, if $\hat{B} = \hat{V}$, then $\mathcal{S}$ sets $\sigma_2 = \xi(V, X)$.
3. *Send*$(\hat{A}, \hat{B}, \mathcal{I}, X, Y, T_B)$: $\mathcal{S}$ executes Step 3 of the protocol. However, if $\hat{A} = \hat{V}$, then $\mathcal{S}$ sets $\sigma_1 = \xi(V, Y)$.
4. *Send*$(\hat{B}, \hat{A}, \mathcal{R}, X, Y, T_B, T_A)$: $\mathcal{S}$ executes Step 4 of the protocol.
5. $\mathcal{H}_2(*)$: $\mathcal{S}$ simulates a random oracle in the usual way.
6. $\mathcal{H}_1(z, *)$: For every new query, $\mathcal{S}$ computes $Z = g^z$. If $Z = V$, then $\mathcal{S}$ aborts and outputs $\mathrm{CDH}(U, V) = U^z$; otherwise $\mathcal{S}$ simulates a random oracle in the usual way.
7. $\mathcal{H}(\hat{A}, \hat{B}, X, Y, \sigma_1, \sigma_2, \sigma_e)$:
   (a) If $\hat{A} = \hat{V}$ and $\sigma_1 \neq \xi(V, Y)$, then $\mathcal{S}$ obtains $\tau_1 = \mathrm{DDH}(V, Y, \sigma_1)$.
      i. If $\tau_1 = 1$, $\mathcal{S}$ returns $\mathcal{H}(\hat{A}, \hat{B}, X, Y, \xi(V, Y), \sigma_2, \sigma_e)$.
      ii. If $\tau_1 = 0$, $\mathcal{S}$ simulates a random oracle in the usual way.
   (b) If $\hat{B} = \hat{V}$ and $\sigma_2 \neq \xi(V, X)$, then $\mathcal{S}$ obtains $\tau_2 = \mathrm{DDH}(V, X, \sigma_2)$.
      i. If $\tau_2 = 1$, $\mathcal{S}$ returns $\mathcal{H}(\hat{A}, \hat{B}, X, Y, \sigma_1, \xi(V, X), \sigma_e)$.
      ii. If $\tau_2 = 0$, $\mathcal{S}$ simulates a random oracle in the usual way.
   (c) $\mathcal{S}$ simulates a random oracle in the usual way.
8. *EphemeralPublicKeyReveal*$(\hat{A})$: $\mathcal{S}$ responds to the query faithfully.
9. *EphemeralKeyReveal*(S): $\mathcal{S}$ responds to the query faithfully.
10. *SessionKeyReveal*(S): $\mathcal{S}$ responds to the query faithfully.

11. *StaticKeyReveal*($\hat{A}$): If $\hat{A} = \hat{V}$, then $\mathcal{S}$ aborts with failure. Otherwise $\mathcal{S}$ responds to the query faithfully.

12. *Establish*($\hat{M}, M$): $\mathcal{S}$ responds to the query faithfully.

13. *Test*(S): $\mathcal{S}$ responds to the query faithfully.

14. $\mathcal{M}$ outputs a guess γ: $\mathcal{S}$ aborts with failure.

Analysis of event $E_1 \wedge M^*$. $\mathcal{S}$'s simulation of $\mathcal{M}$'s environment is perfect except with negligible probability. If $\mathcal{S}$ assigns V to an honest party $\hat{B}$ for whom $\mathcal{M}$ will query $\mathcal{H}_1(v, *)$ without first issuing a *StaticKeyReveal*($\hat{B}$) query, then $\mathcal{S}$ is successful as described in Step 6 and abortions as in Steps 11 and 14 do not occur. Hence, if event $E_1 \wedge M^*$ occurs with probability p_1, then $\mathcal{S}$ is successful with probability $\Pr(S)$ that is bounded by

$$\Pr(S) \geq \frac{1}{\mathbf{n}} p_1. \tag{5.1}$$

Event $E_2 \wedge M^*$. Let T_m be the event "the test session has a matching session owned by an honest party". Event $E_2 \wedge M^*$ is further subdivided into the following complementary events: (i) $E_{2a} = E_2 \wedge M^* \wedge T_m$ and (ii) $E_{2b} = E_2 \wedge M^* \wedge \overline{T_m}$. Let $p_2 = \Pr(E_2 \wedge M^*)$, $p_{2a} = \Pr(E_{2a})$, and $p_{2b} = \Pr(E_{2b})$. Since E_{2a} and E_{2b} are complementary events it follows that $p_2 = p_{2a} + p_{2b}$. Therefore, if event $E_2 \wedge M^*$ occurs with non-negligible probability, then either E_{2a} or E_{2b} occurs with non-negligible probability. The two events are considered separately.

Simulation in event $E_{2a} \wedge M^*$. Suppose that event $E_{2a} \wedge M^*$ occurs with non-negligible probability. $\mathcal{S}$ establishes $\mathbf{n}$ parties that are assigned random static key pairs. $\mathcal{S}$ randomly selects two parties $\hat{C}$, $\hat{D}$ and two integers $i, j \in_R [1, \mathbf{s}]$ subject to the condition that $(\hat{C}, i) \neq (\hat{D}, j)$. $\mathcal{S}$ selects ephemeral key pairs on behalf of honest parties as described in the simulation in event $E_1 \wedge M^*$, with the following exceptions. The ith ephemeral public key selected on behalf of $\hat{C}$ is chosen to be U and the corresponding ephemeral private key is $\tilde{u} \in_R \{0,1\}^\lambda$; note that $\mathcal{S}$ does not know $u = \log_g U = \mathcal{H}_1(c, \tilde{u})$. Similarly, the jth ephemeral public key selected on behalf of $\hat{D}$ is V and the corresponding ephemeral private key is $\tilde{v} \in_R \{0,1\}^\lambda$; $\mathcal{S}$ does not know $v = \log_g V = \mathcal{H}_1(d, \tilde{v})$. The sessions that are subsequently activated with ephemeral public keys U and V are denoted by S^U and S^V, respectively. The simulation of $\mathcal{M}$'s environment proceeds as follows.

1. *Send*($\hat{A}, \tilde{B}$): $\mathcal{S}$ executes Step 1 of the protocol.

2. *Send*$(\tilde{B}, \hat{A}, X)$ issued to $\hat{B}$: $\mathcal{S}$ executes Step 2 of the protocol. If the session created is s^U or s^V, then $\mathcal{S}$ deviates from the protocol description by setting $\sigma_1 = \xi(A, Y)$ and $\sigma_e = \xi(X, Y)$ where $Y \in \{U, V\}$ is the ephemeral public key of the created session.

3. *Send*$(\hat{A}, \hat{B}, \mathcal{I}, X, Y, T_B)$: $\mathcal{S}$ executes Step 3 of the protocol. However, if $X \in \{U, V\}$, then $\mathcal{S}$ deviates by not computing $x = \mathcal{H}_1(a, \tilde{x})$ in Step 3(b) and by setting $\sigma_2 = \xi(B, X)$ and $\sigma_e = \xi(X, Y)$.

4. *Send*$(\hat{B}, \hat{A}, \mathcal{R}, X, Y, T_B, T_A)$: $\mathcal{S}$ executes Step 4 of the protocol.

5. $\mathcal{H}_2(*)$: $\mathcal{S}$ simulates a random oracle in the usual way.

6. $\mathcal{H}_1(a, \tilde{x})$: $\mathcal{S}$ simulates a random oracle in the usual way except if $\tilde{X} = \tilde{u}$ and $\hat{A}$ (the party whose static public key is g^a) is the owner of s^U, or if $\tilde{a} = \tilde{v}$ and $\hat{A}$ is the owner of s^V, in which case $\mathcal{S}$ aborts with failure.

7. $\mathcal{H}(\hat{A}, \hat{B}, X, Y, \sigma_1, \sigma_2, \sigma_e)$:

   (a) If $\{X, Y\} = \{U, V\}$ and $\mathrm{DDH}(X, Y, \sigma_e) = 1$, then $\mathcal{S}$ aborts $\mathcal{M}$ and outputs $\sigma_e = \mathrm{CDH}(U, V)$.

   (b) If $X \in \{U, V\}$ and either $\sigma \neq \xi(B, X)$ or $\sigma_e \neq \xi(X, Y)$, then $\mathcal{S}$ does the following. Using the DDH oracle, set $\tau_2 = 1$ if either $\mathrm{DDH}(B, X, \sigma_2) = 1$ or $\sigma_2 = \xi(B, X)$; otherwise set $\tau_1 = 0$. Similarly, set $\tau_e = 1$ if either $\mathrm{DDH}(X, Y, \sigma_e) = 1$ or $\sigma_e = \xi(X, Y)$, and $\tau_e = 0$ otherwise.

      i. If $\tau_2 = 1$ and $\tau_e = 1$, $\mathcal{S}$ returns $\mathcal{H}(\hat{A}, \hat{B}, X, Y, \sigma_1, \xi(B, X), \xi(X, Y))$.

      ii. If $\tau_2 \neq 1$ or $\tau_e \neq 1$, $\mathcal{S}$ simulates a random oracle in the usual way.

   (c) If $Y \in \{U, V\}$ and either $\sigma_1 \neq \xi(A, Y)$ or $\sigma_e \neq \xi(X, Y)$, then $\mathcal{S}$ does the following. Using the DDH oracle, set $\tau_1 = 1$ if either $\mathrm{DDH}(A, Y, \sigma_1) = 1$ or $\sigma_1 = \xi(A, Y)$; otherwise set $\tau_1 = 0$. Similarly, set $\tau_e = 1$ if either $\mathrm{DDH}(X, Y, \sigma_e) = 1$ or $\sigma_e = \xi(X, Y)$, and $\tau_e = 0$ otherwise.

      i. If $\tau_1 = 1$ and $\tau_e = 1$, $\mathcal{S}$ returns $\mathcal{H}(\hat{A}, \hat{B}, X, Y, \xi(A, Y), \sigma_2, \xi(X, Y))$.

      ii. If $\tau_1 \neq 1$ or $\tau_e \neq 1$, $\mathcal{S}$ simulates a random oracle in the usual way.

   (d) $\mathcal{S}$ simulates a random oracle in the usual way.

8. *EphemeralPublicKeyReveal*$(\hat{A})$: $\mathcal{S}$ responds to the query faithfully.

9. *EphemeralKeyReveal*(s): $\mathcal{S}$ responds to the query faithfully.

10. *SessionKeyReveal*(s): $\mathcal{S}$ responds to the query faithfully.

11. *StaticKeyReveal*($\hat{A}$): $\mathcal{S}$ responds to the query faithfully.

12. *Establish*($\hat{M}, M$): $\mathcal{S}$ responds to the query faithfully.

13. *Test*(S): If the ephemeral public keys used in S are not $\{U, V\}$, then $\mathcal{S}$ aborts with failure. Otherwise $\mathcal{S}$ responds to the query faithfully.

14. $\mathcal{M}$ outputs a guess γ: $\mathcal{S}$ aborts with failure.

Analysis of event $E_{2a} \wedge M^*$. $\mathcal{S}$'s simulation of $\mathcal{M}$'s environment is perfect except with negligible probability. The probability that $\mathcal{M}$ selects S^U or S^V as the test session and the other as its matching session is at least $2/(\mathbf{ns})^2$. Suppose that this is indeed the case (so $\mathcal{S}$ does not abort in Step 13), and also suppose that event E_{2a} occurs. Let $\hat{A}$ and $\hat{B}$ denote the communicating parties for the test session and, without loss of generality, let $\hat{A}$ be the test session owner and U her ephemeral public key. Since $\tilde{u}$ is used only in the test session, $\mathcal{M}$ must obtain it via an *EphemeralKeyReveal* query before making an $\mathcal{H}_1$ query that includes $\tilde{u}$. Similarly, $\mathcal{M}$ must obtain $\tilde{v}$ from the matching session via an *EphemeralKeyReveal* query before making an $\mathcal{H}_1$ query that includes $\tilde{v}$. Under event E_2, the adversary first issues a *StaticKeyReveal* query to a party before making an $\mathcal{H}_1$ query that includes that party's static private key. Since the test session is fresh, $\mathcal{M}$ can query for at most one value in each of the pairs $(a, \tilde{u})$ and $(b, \tilde{v})$; hence $\mathcal{S}$ does not abort as described in Step 6. Under event M^*, except with negligible probability of guessing $\xi(U, V)$, $\mathcal{S}$ is successful as described in Step 7(a) and does not fail as described in Step 14. The success probability of $\mathcal{S}$ is bounded by

$$\Pr(S) \geq \frac{2}{(\mathbf{ns})^2} p_{2a}. \tag{5.2}$$

Simulation in event $E_{2b} \wedge M^*$. Suppose that event $E_{2b} \wedge M^*$ occurs with non-negligible probability. $\mathcal{S}$ establishes $\mathbf{n}$ parties. One of these parties, denoted $\hat{V}$, is selected at random and assigned the static public key V, and $\nu \in_R [1, q-1]$ is used to represent the corresponding static private key. The remaining parties are assigned random static key pairs. Furthermore, $\mathcal{S}$ randomly selects a party $\hat{C}$ and integer $i \in_R [1, \mathbf{s}]$. $\mathcal{S}$ selects ephemeral key pairs on behalf of honest parties as described in the simulation in event $E_1 \wedge M^*$, with the following exception. The ith ephemeral public key selected on behalf of $\hat{C}$ is chosen to be U and the corresponding ephemeral private key is $\tilde{u} \in_R \{0,1\}^\lambda$; note that $\mathcal{S}$ does not know $u = \log_g U = \mathcal{H}_1(c, \tilde{u})$. The session that is subsequently activated with ephemeral public key U is denoted by S^U. The simulation of $\mathcal{M}$'s environment proceeds as follows.

1. *Send*$(\hat{A}, \tilde{B})$: $\mathcal{S}$ executes Step 1 of the protocol.

2. *Send*$(\tilde{B}, \hat{A}, X)$ issued to $\hat{B}$: $\mathcal{S}$ executes Step 2 of the protocol. However, if $\hat{B} = \hat{V}$, then $\mathcal{S}$ deviates from the protocol description by setting $\sigma_2 = \xi(V, X)$. Also, if the session created is S^U, then $\mathcal{S}$ sets $\sigma_1 = \xi(A, B)$ and $\sigma_e = \xi(X, Y)$ where $Y = U$ is the ephemeral public key of the created session.

3. *Send*$(\hat{A}, \hat{B}, \mathcal{I}, X, Y, T_B)$: $\mathcal{S}$ executes Step 3 of the protocol. However, if $\hat{A} = \hat{V}$, then $\mathcal{S}$ deviates by setting $\sigma_1 = \xi(A, Y)$. Also, if $X = U$, then $\mathcal{S}$ deviates by not computing $x = \mathcal{H}_1(a, \tilde{a})$ in Step 3(b) and sets $\sigma_2 = \xi(B, X)$ and $\sigma_e = \xi(X, Y)$.

4. *Send*$(\hat{B}, \hat{A}, \mathcal{R}, X, Y, T_B, T_A)$: $\mathcal{S}$ executes Step 4 of the protocol.

5. $\mathcal{H}_2(*)$: $\mathcal{S}$ simulates a random oracle in the usual way.

6. $\mathcal{H}_1(a, \tilde{x})$: $\mathcal{S}$ simulates a random oracle in the usual way except if $\tilde{x} = \tilde{u}$ and $\hat{A}$ (the party whose static public key is g^a) is the owner of S^U, in which case $\mathcal{S}$ aborts with failure.

7. $\mathcal{H}(\hat{A}, \hat{B}, X, Y, \sigma_1, \sigma_2, \sigma_e)$:

   (a) If $\hat{A} = \hat{V}$ and $\sigma_1 \neq \xi(V, Y)$, $\mathcal{S}$ obtains $\tau_1 = \mathrm{DDH}(V, Y, \sigma_1)$.

      i. If $\tau_1 = 1$ and $Y = U$, $\mathcal{S}$ aborts $\mathcal{M}$ and outputs $\mathrm{CDH}(U, V) = \sigma_1$.

      ii. If $\tau_1 = 1$ and $Y \neq U$, $\mathcal{S}$ returns $\mathcal{H}(\hat{A}, \hat{B}, X, Y, \xi(V, Y), \sigma_2, \sigma_e)$

      iii. If $\tau_1 = 0$, $\mathcal{S}$ simulates a random oracle in the usual way.

   (b) If $\hat{B} = \hat{V}$ and $\sigma_2 \neq \xi(V, X)$, $\mathcal{S}$ obtains $\tau_2 = \mathrm{DDH}(V, X, \sigma_2)$.

      i. If $\tau_2 = 1$ and $X = U$, $\mathcal{S}$ aborts $\mathcal{M}$ and outputs $\mathrm{CDH}(U, V) = \sigma_2$.

      ii. If $\tau_2 = 1$ and $X \neq U$, $\mathcal{S}$ returns $\mathcal{H}(\hat{A}, \hat{B}, X, Y, \sigma_1, \xi(V, X), \sigma_e)$.

      iii. If $\tau_2 = 0$, $\mathcal{S}$ simulates a random oracle in the usual way.

   (c) $\mathcal{S}$ simulates a random oracle in the usual way.

8. *EphemeralPublicKeyReveal*$(\hat{A})$: $\mathcal{S}$ responds to the query faithfully.

9. *EphemeralKeyReveal*(S): $\mathcal{S}$ responds to the query faithfully.

10. *SessionKeyReveal*(S): $\mathcal{S}$ responds to the query faithfully.

11. *StaticKeyReveal*$(\hat{A})$: $\mathcal{S}$ responds to the query faithfully unless $\hat{A} = \hat{V}$ in which case $\mathcal{S}$ aborts with failure.

12. *Establish*$(\hat{M}, M)$: $\mathcal{S}$ responds to the query faithfully.

13. *Test*(s): If the peer of s is not $\hat{V}$ or the outgoing ephemeral public key is not U, then $\mathcal{S}$ aborts with failure. Otherwise $\mathcal{S}$ responds to the query faithfully.

14. $\mathcal{M}$ outputs a guess γ: $\mathcal{S}$ aborts with failure.

Analysis of event $E_{2b} \wedge M^*$. The probability that the test session has peer $\hat{V}$ and outgoing ephemeral public key U is at least $1/(\mathbf{n}^2\mathbf{s})$. Suppose that this is indeed the case (so $\mathcal{S}$ does not abort in Step 13), and suppose that event E_{2b} occurs. Let $\hat{A}$ be the owner of the test session s^U. The experiment may fail if $\mathcal{M}$ queries $\mathcal{H}_1$ with $(a, \tilde{u})$. Since $\tilde{u}$ is used only in the test session, $\mathcal{M}$ must obtain it via an *EphemeralKeyReveal* query before making an $\mathcal{H}_1$ query that includes $\tilde{u}$. Under event E_2, the adversary first issues a *StaticKeyReveal* query to a party before making an $\mathcal{H}_1$ query that includes that party's static private key. Since the test session is fresh, $\mathcal{M}$ can query for at most one value in the pair $(a, \tilde{u})$; hence $\mathcal{S}$ does not abort as described in Step 6. Since the test session is fresh, $\mathcal{M}$ is not allowed to issue a *StaticKeyReveal* query to $\hat{V}$; hence $\mathcal{S}$ does not abort as described in Step 11.

Now, $\hat{A}$ receives an incoming tag T_V before the test session completes. In event $\overline{T_m}$ the test session has no matching session. Since key confirmation tags are not repeated except with negligible probability, $\mathcal{M}$ must have computed tag T_V. Since T_V is an output of a random oracle $\mathcal{H}_2$, $\mathcal{M}$ must have obtained the input κ_m by querying $\mathcal{H}$ with $\mathrm{CDH}(U, V)$. In this case, $\mathcal{S}$ is successful as described in Step 7 and does not abort in Step 14.

The success probability of $\mathcal{S}$ is bounded by

$$\Pr(S) \geq \frac{1}{(\mathbf{n}^2\mathbf{s})} p_{2b}. \tag{5.3}$$

Overall analysis. Suppose that event M occurs and hence $\Pr(M^*) \geq p(\lambda)$. Combining the results from Equations 5.1, 5.2 and 5.3 the success probability of $\mathcal{S}$ is

$$\Pr(\mathcal{S}) \geq \max\left\{ \frac{1}{\mathbf{n}} p_1,\ \frac{2}{(\mathbf{ns})^2} p_{2a},\ \frac{1}{\mathbf{n}^2\mathbf{s}} p_{2b} \right\}, \tag{5.4}$$

which is non-negligible in λ.

The simulation requires $\mathcal{S}$ to perform group exponentiations, access the DDH oracle, and simulate random oracles. Since $q = \Theta(2^\lambda)$, a group exponentiation takes time $\mathcal{T}_\mathcal{G} = O(\lambda)$ group multiplications and assume that a DDH oracle call takes time $\mathcal{T}_{\mathrm{DDH}} = O(\lambda)$. Responding to an $\mathcal{H}$ query takes time $\mathcal{T}_\mathcal{H} = O(\lambda)$; similarly responding to $\mathcal{H}_1$ and $\mathcal{H}_2$ queries takes time $\mathcal{T}_{\mathcal{H}_1}(\lambda)$ and $\mathcal{T}_{\mathcal{H}_2}(\lambda)$, respectively.

Taking the largest times from among all simulations for answering $\mathcal{M}$'s query, the running time of $\mathcal{S}$ is bounded by

$$\mathcal{T}_{\mathcal{S}} \leq (5\mathcal{T}_{\mathcal{G}} + 3\mathcal{T}_{\mathrm{DDH}} + \mathcal{T}_{\mathcal{H}} + \mathcal{T}_{\mathcal{H}_1} + \mathcal{T}_{\mathcal{H}_2})\,\mathcal{T}_{\mathcal{M}}. \tag{5.5}$$

Thus, if $\mathcal{M}$ is polynomially bounded, then there is an algorithm $\mathcal{S}$ that succeeds in solving the GDH problem in $\mathcal{G}$ with non-negligible probability. Furthermore $\mathcal{S}$ runs in polynomial time, contradicting the GDH assumption in $\mathcal{G}$. This concludes the proof of Theorem 5.5.1. □

5.6 Concluding remarks

By comparing the CK01 and the CK02 models for key agreement, this chapter demonstrates that security in one model does not guarantee security or even executability in the other model. §5.4.3 presented a combined security model and definition, that simultaneously encompasses strengthened versions of the Canetti-Krawczyk definitions. The new definition is stronger in that it permits the adversary to learn ephemeral public keys before they are used, and to learn secret information from the session being attacked. Useful directions for future research would be the development of an optimized protocol that satisfies the new security definition, perhaps modified to allow for identity concealment and the extension of the definition to capture a wider class of key agreement protocols.

Chapter 6

Efficiency considerations

6.1 Motivation

Until now in all the protocols and security arguments that were presented, two important assumptions were not highlighted. Both assumptions have impact on efficiency, which is an important key establishment attribute.

The first assumption concerns the *lifetime* of ephemeral public keys. Until now, an ephemeral public key is used in only one session. However, selecting an ephemeral private key and computing the corresponding ephemeral public key requires a group exponentiation. A party that is under heavy load may be inclined to reuse an ephemeral public key across multiple sessions, to improve performance. This is especially likely to occur if the protocol specifications do not explicitly require ephemeral keys to be used only in one session.

The second assumption relates to *public-key validation*. Public key validation is the process whereby a party verifies that an incoming message satisfies certain properties. The main focus will be on validating that ephemeral public keys belong to the underlying group $\mathcal{G}$ where the protocol is supposed to be executed. Depending on $\mathcal{G}$, public-key validation may be an essentially free operation or it may be equivalent to a full group exponentiation. Thus, in some important practical scenarios, public-key validation has impact on the efficiency of the key agreement protocol.

The motives and reasons behind these assumptions are analysed in this chapter.

6.2 Review

Public-key validation is a sound cryptographic practice. In [49], *small-subgroup attacks* demonstrated the importance of public-key validation in Diffie-Hellman type key agreement protocols. In [12, 4], *invalid-curve attacks* were designed that are effective on some elliptic curve protocols if the receiver of a point does not verify that the point indeed lies on the chosen elliptic curve. Kunz-Jacques et al. [43] showed that the zero-knowledge proof proposed in [5] for proving possession of discrete logarithms in groups of unknown order can be broken if a dishonest verifier selects invalid parameters during its interaction with the prover. More recently, Chen, Cheng and Smart [22] illustrated the importance of public-key validation in identity-based key agreement protocols that use bilinear pairings.

6.2.1 Terminology and notation

Assume that there is an algebraic structure R (e.g., a field, ring, or group) such that:

1. The elements of R are represented in the same format as elements of $\mathcal{G}$ (e.g., bitstrings of the same length).
2. The group operation for $\mathcal{G}$ is defined on elements of R.
3. There is a subset $\mathcal{G}'$ of R such that $\mathcal{G}'$ is a cyclic group of order t, with respect to the operation defined on $\mathcal{G}$; typically γ will denote the generator of $\mathcal{G}'$.

For the remainder of this chapter, t will be 2^r for some small r (e.g., $r = 4$) or t will be small enough so that an adversary can feasibly perform t operations (e.g., $t < 2^{40}$).

6.2.2 Small subgroup attacks

Suppose that in sDH (see Figure 1.2), neither $\hat{A}$ nor the certification authority verify that static public keys are non-identity elements of $\mathcal{G}$. Following the notation in §6.2.1 the adversary who controls party $\hat{M}$, selects an invalid static public key $M = \gamma$. Upon receiving M, party $\hat{A}$ computes $\kappa = \mathcal{H}(\sigma_s)$ where $\sigma_s = M^a = \gamma^a$. Suppose now that $\hat{A}$ subsequently sends $\hat{M}$ an authenticated message $(mesg, T_A)$ where $T_A = \mathrm{MAC}_\kappa(mesg)$. Then $\mathcal{M}$ iteratively computes $\kappa' = \mathcal{H}(\gamma^c)$ and $T'_A = \mathrm{MAC}_{\kappa'}(mesg)$ for $c = 0, 1, 2, \ldots$ until $T'_A = T_A$, in which case $c = a \bmod t$ with high probability. By repeating this procedure for several values of t that are pairwise

relatively prime (and possibly different groups $\mathcal{G}'$), the adversary can efficiently determine $\hat{A}$'s static private key a by the Chinese Remainder Theorem.

6.2.3 DSA-type groups

A DSA-type group $\mathcal{G}$ is the subgroup of prime order q of the multiplicative group $\mathcal{G}'$ of a prime field $\mathbb{Z}_p$. The bitlength of q is substantially smaller than that of p. For example, the bitlengths of q and p may be 160 and 1024 respectively, or 256 and 3072 respectively. Since the group parameters are typically generated by randomly selecting a prime q, and then randomly selecting even integers k of the appropriate bitlength until $p = 1 + kq$ is prime, then with non-negligible probability (see [6]), $(p-1)/q$ will have a smooth divisor greater than q. Hence DSA-type groups are vulnerable to small-subgroup attacks if the recipient of a public key M does not verify that $M \in \mathcal{G} \setminus \{1\}$; this validation can be accomplished by checking that M is an integer in the interval $[2, p-1]$ and that $M^q = 1$. Another countermeasure against these attacks is to select domain parameters p and q so that $(p-1)/q$ does not have any small odd prime factor.

6.2.4 Safe prime groups

In these groups, $\mathcal{G}$ is the subgroup of prime order q of the multiplicative group of a prime field $\mathbb{Z}_p$, where $p = 2q + 1$. If $\hat{A}$ checks that the public key M is an integer in the interval $[2, p-2]$, then, since the multiplicative group of $\mathbb{Z}_p$ has order $2q$, attacks like the ones described above can reveal at most a single bit of a. Thus the small-subgroup attacks, described in §6.2.2, are not effective in the case of safe prime groups.

6.2.5 Elliptic curve groups

Let $E : V^2 = U^3 + \alpha U + \beta$ be an elliptic curve of prime order q defined over a prime field $\mathbb{F}_p$, and let $\mathcal{G} = E(\mathbb{F}_p)$. In [12] (see also [4] and §6.8.2) it was observed that the usual formulae for adding points in $E(\mathbb{F}_p)$ do not explicitly depend on the coefficient β. Hence $\mathcal{M}$ could launch invalid curve attacks similar to the small-subgroup attack described above by selecting $M \in \mathcal{G}' = E'(\mathbb{F}_p)$, where $E' : V^2 = U^3 + \alpha U + \beta'$ is an elliptic curve whose order is divisible by a relatively small prime factor t. Party $\hat{A}$ can easily thwart the attack by checking that M is a point in $E(\mathbb{F}_p)$ (and is not the point at infinity) or employing point compression techniques.

6.3 MQV and HMQV review

The MQV protocols [47] are a family of authenticated Diffie-Hellman protocols that have been widely standardized [2, 3, 35, 71]. In the two-pass and three-pass versions of the protocol, the communicating parties $\hat{A}$ and $\hat{B}$ exchange static and ephemeral public keys, and thereafter derive a secret key from these values. In the one-pass version, only one party contributes an ephemeral public key. In 2005, Krawczyk [41] presented the HMQV protocols, which are hashed variants of the MQV protocols. The primary advantages of HMQV over MQV are better performance and a rigorous security proof. The improved performance of HMQV is a direct consequence of not requiring the validation of ephemeral and static public keys — unlike with MQV where these operations are mandated. Despite the omission of public-key validation, Krawczyk was able to devise proofs that the HMQV protocols are secure in the random oracle model assuming the intractability of the computational Diffie-Hellman problem (and some variants thereof) in the underlying group.

6.3.1 HMQV description

In the following $\mathcal{H}_2$ is an l-bit hash function where $l = (\lfloor \log_2 q \rfloor + 1)/2$.

In the two-pass HMQV protocol as presented in [41], parties $\hat{A}$ and $\hat{B}$ establish a secret session key as follows:

1. $\hat{A}$ selects an ephemeral private key $x \in_R [0, q-1]$ and computes her ephemeral public key $X = g^x$. $\hat{A}$ then sends $(\hat{A}, \hat{B}, X)$ to $\hat{B}$.

2. Upon receiving $(\hat{A}, \hat{B}, X)$, $\hat{B}$ checks that $X \neq 0$. Note that $0 \notin \mathcal{G}$. The check $X \neq 0$ and $Y \neq 0$ makes sense in some settings, e.g., when $\mathcal{G}$ is a multiplicative subgroup of a finite field; in this case 0 is the additive identity of the field.

   Party $\hat{B}$ selects an ephemeral key pair (y, Y), and sends $(\hat{B}, \hat{A}, Y)$ to $\hat{A}$. $\hat{B}$ proceeds to compute $s_B = y + \mathrm{E}b \bmod q$ and $\sigma = (XA^{\mathrm{D}})^{s_B}$ where $\mathrm{D} = \mathcal{H}_2(X, \hat{B})$ and $\mathrm{E} = \mathcal{H}_2(Y, \hat{A})$. The HMQV paper [41] does not explicitly state that s_A and s_B should be computed modulo q. The attacks described later can still be launched if s_A and s_B are not reduced modulo q.

3. Upon receiving $(\hat{B}, \hat{A}, Y)$, $\hat{A}$ checks that $Y \neq 0$, and computes $s_A = x + \mathrm{D}a \bmod q$ and $\sigma = (YB^{\mathrm{E}})^{s_A}$ where again $\mathrm{D} = \mathcal{H}_2(X, \hat{B})$ and $\mathrm{E} = \mathcal{H}_2(Y, \hat{A})$.

4. The secret session key is $\kappa = \mathcal{H}(\sigma) = \mathcal{H}(g^{s_A s_B})$.

The messages transmitted in Steps (1) and (2) may include certificates for the static public keys A and B, respectively. Note that HMQV does not mandate that static and ephemeral public keys be validated, i.e., verified as being non-identity elements of $\mathcal{G}$. Krawczyk [41] provided a very extensive analysis of HMQV. He proved that the protocol is CK01-secure under the assumptions that $\mathcal{H}$ and $\mathcal{H}_2$ are random oracles and that the CDH problem in $\mathcal{G}$ is intractable. Krawczyk also proved that the protocol is resistant to attacks when the adversary is permitted to learn the ephemeral private keys of sessions; for this property the GDH and KEA1 assumptions in $\mathcal{G}$ are needed.

6.3.2 MQV description

The three essential differences between the MQV protocol (as standardized in [71]) and the HMQV protocol are the following:

1. Static and ephemeral public keys must be validated in MQV. Actually 'embedded' validation may be performed on ephemeral public keys; the details are not relevant to the attacks presented in this chapter.
2. In MQV, the integers D and E are derived from the group elements X and Y, respectively. For example, if $\mathcal{G}$ is a group of elliptic curve points, then D and E are derived from the l least significant bits of the x-coordinates of X and Y respectively.
3. The secret session key is $\kappa = \mathcal{H}(\sigma, \hat{A}, \hat{B})$. The identities $\hat{A}$, $\hat{B}$ are included in the derivation of κ from σ in order to thwart Kaliski's unknown-key share attack [18].

6.4 Reusing ephemeral keys

A party may choose to reuse ephemeral public keys in a Diffie-Hellman key agreement protocol in order to reduce its computational workload or to mitigate against denial-of-service attacks. As shown next, reusing ephemeral keys if domain parameters are not appropriately selected or public keys are not appropriately validated may affect the security of the underlying protocol.

Some Diffie-Hellman protocols implicitly or explicitly require that an ephemeral key pair can be used in only one session. For example, the ANSI X9.42 [2]

standard which specifies several Diffie-Hellman protocols states that an ephemeral key is a "private or public key that is unique for each execution of a cryptographic scheme. An ephemeral private key is to be destroyed as soon as computational need for it is complete." Other protocols do not place any restrictions on the reuse of ephemeral keys. For example, the SIGMA protocol [40], which is the basis for the signature-based modes of the IKE (versions 1 and 2) protocols, allows for the reuse of an ephemeral public key "by the same party across different sessions", and this advice is followed in the IKEv2 standard [27, §2.12].

The primary reason for reusing ephemeral public keys is to increase efficiency by reducing the number of costly exponentiations a party has to perform. Another reason is to mitigate against denial-of-service (DoS) attacks. For example the JFKi protocol [1] allows a responder to reuse an ephemeral public key, whereby the responder does not have to do any expensive computations in its first response and only performs costly exponentiations after receiving the second message from the session's initiator. Since the initiator has to perform costly cryptographic operations when preparing the second message, this protocol offers the responder some resistance to DoS attacks. The authors of [1] go on to say that it is inadvisable *not* to reuse ephemeral public keys "in times of high load (or attack)."

6.4.1 S/MIME

S/MIME (version 3.1) is an IETF standard for securing email [26, 34, 33]. S/MIME can be used to provide several security services including confidentiality, data integrity, data origin authentication, and non-repudiation, and allows the use of various public-key cryptographic algorithms including RSA, DSA and Diffie-Hellman. This section examines the vulnerability to small-subgroup attacks of an encryption-only mode in S/MIME (called "Enveloped-only" in [26]) when ephemeral-static Diffie-Hellman (as described in [64]) is used for key agreement.

The following describes the S/MIME encryption-only mode. Let $\mathcal{G}$ be a DSA-type group. Let Enc and Dec denote the encryption and decryption functions for a symmetric-key encryption scheme such as Triple-DES. Party $\hat{A}$ encrypts an email msg for $\hat{B}$ as follows:

1. Obtain an authentic copy of $\hat{B}$'s static public key B.
2. Select an ephemeral private key $x \in_R [1, q-1]$ and compute the ephemeral public key X and the session key $\kappa = \mathcal{H}(\sigma)$ where $\sigma = B^x = g^{bx}$.

3. Select at random a content-encryption key κ_{ce} for the symmetric-key encryption scheme.

4. Compute a 64-bit checksum v for κ_{ce}, and compute $c_1 = \text{Enc}_\kappa(\kappa_{ce} \mid v)$. More precisely, the checksum v consists of the 64 most significant bits of SHA-1(κ_{ce}), and $\kappa_{ce} \mid v$ is encrypted twice in succession using the CBC mode of encryption [32].

5. Compute $c_2 = \text{Enc}_{\kappa_{ce}}(msg)$.

6. Send (X, c_1, c_2) to $\hat{B}$.

Upon receiving (X, c_1, c_2), party $\hat{B}$ computes $\kappa = \mathcal{H}(X^b)$, decrypts c_1, and verifies that the checksum for the recovered key κ_{ce} is correct. If so, then $\hat{B}$ decrypts c_2 to obtain msg.

The S/MIME standards allow a party $\hat{A}$ to reuse an ephemeral key pair (x, X) for an unspecified period of time. For example, [64, §2.3] (see also [33, §4.1.1]) advises that some additional data, such as a counter, be appended to σ prior to hashing to ensure that a different session key is generated even if the ephemeral key is being reused: "If, however, the same ephemeral sender key is used for multiple messages (e.g. it is cached as a performance optimization) then a separate partyAInfo MUST be used for each message." However, while [64] recommends that the recipient $\hat{B}$ validate the received public key X in order to protect against small-subgroup attacks that aim to learn its static private key b, there is no requirement that the sender $\hat{A}$ validate the recipient's static public key B; [75, §2.1] explains that this protection is not necessary because a dishonest party $\hat{B}$ who learns x via a small-subgroup attack isn't able to mount any interesting attacks because "the key is ephemeral and only associated with a message that the recipient can already decrypt..."

Observe that an interesting small-subgroup attack *can* be mounted in the situation where $\hat{A}$ reuses (x, X) to encrypt different messages to more than one party. A dishonest party $\hat{M}$ chooses its static public key $M = \gamma$ to be an element of small order t in $\mathbb{Z}_p^*$. Upon receiving (X, c_1, c_2), $\hat{M}$ iteratively computes $\kappa' = \mathcal{H}(\gamma^c)$ and decrypts c_1 using κ' for $c = 0, 1, 2, \ldots$ until the checksum verifies. Party $\hat{M}$ then learns that $c = x \bmod t$, and can repeat this procedure for pairwise relatively prime t to recover x. Hereafter, $\hat{M}$ is able to decrypt messages that $\hat{A}$ encrypts using x for other (honest) parties.

As mentioned in §6.2.3 such attacks can be thwarted by validating static public keys, or by selecting DSA parameters p, q so that $(p-1)/q$ does not have any small odd prime factors.

6.4.2 HMQV

The HMQV security proof assumes that ephemeral keys are used only once, and that ephemeral private keys are securely destroyed after they have been used. However, in practice it may be tempting (in the absence of explicit warnings) for an implementer to allow for the reuse of ephemeral public keys in order to improve performance. If party $\hat{A}$ decides to reuse an ephemeral key pair (x, X), then an adversary $\mathcal{M}$ who controls party $\hat{M}$ may be able to mount a devastating attack — one that reveals $\hat{A}$'s static private key a.

The attack assumes DSA-type groups where $(p-1)/q$ has several small (e.g. less than 2^{40}) pairwise relatively prime factors whose product is greater than q; let t be one such factor. The adversary obtains a certificate for $\hat{M}$'s valid static public key $M = g^m$. Upon receiving (A, X) from $\hat{A}$, the attacker selects an ephemeral public key $Z \in \mathcal{G}'$ of order t, and sends (M, Z) to $\hat{A}$. The public keys M and Z satisfy the HMQV checks, so $\hat{A}$ computes $\sigma = (ZM^{\mathrm{E}})^s$ and $\kappa = \mathcal{H}(\sigma)$ where $s = x + \mathrm{D}a \bmod q$, $\mathrm{D} = \mathcal{H}_2(X, \hat{M})$, and $\mathrm{E} = \mathcal{H}_2(Z, \hat{A})$. Suppose now that $\hat{A}$ sends $\hat{M}$ an authenticated message $(msg, T_A = \mathrm{MAC}_\kappa(msg))$. Note that

$$\sigma = (ZM^{\mathrm{E}})^s = Z^s M^{\mathrm{E}s} = Z^s (g^s)^{\mathrm{E}m} = Z^s (XA^{\mathrm{D}})^{\mathrm{E}m}.$$

Hence $\mathcal{M}$ iteratively computes $\kappa' = \mathcal{H}(Z^c (XA^{\mathrm{D}})^{\mathrm{E}m})$ and $T'_A = \mathrm{MAC}_{\kappa'}(msg)$ for $c = 0, 1, 2, \ldots$ until $T'_A = T_A$ in which case $c = s \bmod t$. After repeating this procedure for several pairwise relatively prime orders t, $\hat{M}$ can determine s by the Chinese Remainder Theorem.

The adversary now repeats the attack using a different identifier $\hat{M}'$ (or perhaps by colluding with a third party $\hat{C}$). Consequently adversary learns $s' = x + \mathrm{D}'a \bmod q$, where $\mathrm{D}' = \mathcal{H}_2(X, \hat{M}')$. Since $\mathrm{D}' \neq \mathrm{D}$ with very high probability, the adversary can then compute $a = (s - s')(\mathrm{D} - \mathrm{D}')^{-1} \bmod q$.

The attack is possible because $\hat{A}$ is not required to validate public keys. Omission of validation is intentional in HMQV in order to improve performance — the only checks required on B and Y are that they belong to $\mathcal{G}' \setminus \{1\}$. However, the attack *cannot* be mounted on the version of HMQV as described in [42], since that version of the protocol assumes that ephemeral public keys are never reused. Rather, the attack highlights the danger of reusing ephemeral public keys in a key agreement protocol for which the security analysis assumed that ephemeral public keys are never reused.

6.4.3 Standards' requirements

As mentioned in §6.4, the IKEv2 protocol [27] allows for the reuse of ephemeral private keys. However, the protocol is immune to the small-subgroup attack described in §6.4.1 because IKEv2 requires the use of safe prime groups [66]. More recently, elliptic curve groups have been proposed for IKE and IKEv2 [30, 17]. To prevent invalid-curve attacks analogous to the small-subgroup attack described in §6.4.1, a sender who reuses an ephemeral key pair to encrypt different messages for more than one party should validate the recipients' static public keys.

The recent NIST SP800-56A [71] standard for key agreement explicitly disallows the reuse of ephemeral keys with one exception — a sender may reuse an ephemeral key if the resulting session key is used to transport the same keying material, and if all these transactions occur "simultaneously" (or within a short period of time). Since SP 800-56A mandates that all public keys be validated, this reuse of ephemeral public keys appears to be sound. It would be a useful exercise to formally verify the belief that reusing ephemeral keys as allowed in SP800-56A does not introduce any security weaknesses.

6.5 On public-key validation

Menezes [52] identified some flaws in the HMQV security proofs and presented small-subgroup attacks on the protocols. The attacks exploit the omission of validation for both ephemeral and static public keys, and allow an adversary to recover the victim's static private key. The attacks on the one-pass protocol are the most realistic, while the attacks on the two-pass and three-pass protocols are harder to mount in practice because the adversary needs to learn some of the victim's ephemeral private keys.

Subsequently, §6.6–§6.11 further investigate the effects of omitting public-key validation in HMQV and MQV, by focusing on the two-pass HMQV protocol (called *the* HMQV protocol from here on), which is the core member of the HMQV family. §6.7.3 identifies a subtle flaw in the HMQV security proof which leads to an attack that does not require knowledge of ephemeral private keys, thereby contradicting the claim made in [41] that the HMQV protocol (without public-key validation) is provably secure if the adversary never learns any ephemeral private keys. The vulnerability of HMQV and MQV if only static public keys are validated, or if only ephemeral public keys are validated are considered separately. These hypothetical scenarios are worth investigating because the reasons for omit-

ting public-key validation can be different for ephemeral and static keys — validation of ephemeral public keys may be omitted for performance reasons, while validation of static public keys may be omitted because the certification authority may not be configured to perform such tests [41].

Many of the attacks described later cannot be mounted in realistic settings. For example, the aforementioned attack on HMQV that does not require knowledge of ephemeral private keys is described in certain underlying groups that have never been proposed for practical use. Moreover, this attack fails if the underlying group is a DSA-like group or a prime-order subgroup of an elliptic curve group as proposed for standardization in [42]. The attacks based on validation omission do not necessarily imply that (fully) validated public keys are a must in all Diffie-Hellman key agreement protocols. For example, the version of HMQV proposed in [42] only requires that a few simple and efficient checks be performed on static and ephemeral public keys. Moreover, even in the situation where one is concerned that ephemeral private keys might be leaked, [42] only requires that ephemeral and static public keys be *jointly* validated, thus saving a potentially expensive validation step (cf. §6.10).

6.6 Validation and ephemeral private-key leakage

The following attack on HMQV was presented in [52]. The attack exploits the omission of public-key validation for ephemeral and static public keys, and also the ability of the adversary to learn the victim's ephemeral private keys.

Under the conditions in §6.2.1 the attack proceeds as follows. The adversary $\mathcal{M}$, who controls $\hat{M}$, chooses an element $\gamma \in \mathcal{G}'$ of order $t = 2^r$, selects $m, z \in [1, t-1]$, computes $M = \gamma^m$ and $Z = \gamma^z$, and sends $(\hat{M}, \hat{B}, Z)$ to $\hat{B}$. While $\hat{B}$ is computing the session key $\kappa = \mathcal{H}((ZM^{\mathrm{D}})^{s_B})$, the adversary learns $\hat{B}$'s ephemeral private key y. Let $\beta = ZM^{\mathrm{D}} = \gamma^{z+\mathrm{D}m}$, so $\kappa = \mathcal{H}(\beta^{s_B})$. The adversary then learns the session key κ. Suppose, for example, that $\hat{B}$ sends $\hat{A}$ an authenticated message $(mesg, T_B = \mathrm{MAC}_\kappa(mesg))$. Then $\mathcal{M}$ can learn κ by computing $T'_B = \mathrm{MAC}_{\kappa'}(mesg)$ where $\kappa' = \mathcal{H}(\beta^c)$ for $c = 0, 1, 2, \ldots, t-1$ until $T'_B = T_B$, in which case $\hat{M}$ has determined $s_B \bmod t$. After repeating this procedure a few times, $\hat{M}$ can use the Leadbitter-Smart lattice attack [48] to find the l most significant bits of b. The remaining l bits of b can thereafter be determined in $O(q^{1/4})$ time using Pollard's lambda method [63].

6.7 Validation and no ephemeral private-key leakage

The adversary in the attack of §6.6 requires knowledge of the victim's ephemeral private keys. While resistance to ephemeral private key leakage is a desirable attribute of a key establishment protocol — in [41] resistance of Diffie-Hellman protocols to damage from the disclosure of ephemeral private keys is described as a 'prime security concern' — it is arguably not a fundamental security requirement. In [41] it is claimed that the HMQV protocol is *provably secure* if the adversary does not learn any ephemeral private keys. The following subsection presents an attack refuting that claim.

6.7.1 A new attack

Extending the terminology from §6.2.1, suppose that the CDH problem in $\mathcal{G}$ is intractable and that R is a ring such that:

1. The elements of R are represented in the same format as elements of $\mathcal{G}$ (e.g., bitstrings of the same length).
2. The multiplication operation for R is defined in the same way as the operation for $\mathcal{G}$. In particular, $\mathcal{G}$ is a subgroup of the group of units $U(R)$ of R.
3. There exists an element $Z \in R$, $Z \neq 0$, such that $Z^2 = 0$ (where "0" denotes the additive identity element in R).

The new attack on HMQV assumes that parties do not validate ephemeral public keys. The adversary $\mathcal{M}$ intercepts the message $(\hat{A}, \hat{B}, X)$ sent by $\hat{A}$ and replaces it with $(\hat{A}, \hat{B}, Z)$. Similarly, $\mathcal{M}$ intercepts $\hat{B}$'s response $(\hat{B}, \hat{A}, Y)$ and forwards $(\hat{B}, \hat{A}, Z)$ to $\hat{A}$. If R is commutative, then, assuming that $s_A \geq 2$ and $s_B \geq 2$, it is easy to see that both $\hat{A}$ and $\hat{B}$ compute the session key $\kappa = \mathcal{H}(0)$. Of course, $\mathcal{M}$ can also compute this key.

If R is not commutative, then the value of the session key depends on the particular exponentiation method used by the parties. Suppose that $\hat{A}$ determines the session key by first calculating $t_A = es_A \bmod q$ and then using simultaneous multiple exponentiation [55, Algorithm 14.88] to compute $\sigma = Z^{s_A} B^{t_A}$ and $\kappa = \mathcal{H}(\sigma)$. This algorithm first computes ZB and initializes an accumulator to 1. It then repeatedly examines the bits of s_A and t_A from left to right. During each iteration, either 1, Z, B or ZB is multiplied into the accumulator which is then squared.

Now, if the most significant bits of s_A and t_A are 1 and 0, respectively, then the accumulator takes on the values 1, Z, $Z^2, \ldots$. Hence $\hat{A}$ computes $\sigma = 0$. Similarly, $\hat{B}$ may compute $\sigma = 0$, in which case $\mathcal{M}$ also learns the session key $\kappa = \mathcal{H}(0)$.

6.7.2 Supporting examples

The following two examples are of groups that satisfy the conditions of §6.7.1. These examples do not have any immediate practical relevance since such groups are not being deployed in practice. Nonetheless, they serve to refute the claim made in [41] that HMQV is provably secure regardless of the representation used for the elements of the $\mathcal{G}$ (subject to the constraint that the CDH problem in $\mathcal{G}$ be intractable).

A commutative example. Let p be a 1024-bit prime such that $p - 1$ has a 160-bit prime divisor q. Consider the commutative ring $R = \mathbb{Z}_{p^2}$. Then $U(R)$ is cyclic and $\#U(R) = p(p-1)$. Let $\mathcal{G}$ be the order-q cyclic subgroup of $U(R)$. The CDH problem in $\mathcal{G}$ is believed to be intractable. The element $Z = p \in \mathbb{Z}_{p^2}$ satisfies $Z \neq 0$ and $Z^2 = 0$.

A non-commutative example. Again, let p be a 1024-bit prime such that $p - 1$ has a 160-bit prime divisor q. Consider the non-commutative ring R of 2×2 matrices over $\mathbb{F}_p$. Then $\#U(R) = (p^2 - 1)(p^2 - p)$. Let $g \in U(R)$ be an element of order q, and let $\mathcal{G} = \langle g \rangle$. The CDH problem in $\mathcal{G}$ is equivalent to the CDH problem in the order-q subgroup of $\mathbb{F}_p^*$ (see [56]) and is therefore believed to be intractable. The element

$$Z = \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix}$$

satisfies $Z \neq 0$ and $Z^2 = 0$.

6.7.3 Flaw in the HMQV proof

The HMQV security proof in [41] has two main steps. First, an 'exponential challenge-response' signature scheme XCR is defined and proven secure in the random oracle model under the assumption that the CDH problem in $\mathcal{G}$ is intractable. Second, the security of XCR (actually a 'dual' version of XCR) is proven to imply the security of HMQV.

In the XCR signature scheme, a verifier $\hat{A}$ selects $x \in_R [0, q-1]$ and sends the challenge $X = g^x$ and a message m to the signer $\hat{B}$. $\hat{B}$ responds by selecting $y \in_R [0, q-1]$ and sending the signature $(Y = g^b, \sigma = X^{s_B})$ to $\hat{A}$ where $s_B = y + \mathrm{E}b \bmod q$, $\mathrm{E} = \mathcal{H}_2(Y, m)$, and b, B is $\hat{B}$'s static key pair. The signature is accepted by $\hat{A}$ provided that $Y \neq 0$ and $\sigma = (YB^{\mathrm{E}})^x$. XCR signatures are different from ordinary digital signatures — $\hat{A}$ cannot convince a third party that $\hat{B}$ generated a signature (Y, σ) for message m and challenge X because $\hat{A}$ could have generated this signature herself.

The XCR security proof in [41] uses the forking lemma of Pointcheval and Stern [62]. The proof hypothesizes the existence of a forger who, on input $B, X_0 \in_R \mathcal{G}$ and a signing oracle for $\hat{B}$, produces a message m_0 and a valid signature (Y_0, σ) for m_0, so that $Y_0 \neq 0$ and $\sigma = (Y_0 B^{\mathrm{E}})^{a_0}$ where $\mathrm{E} = \mathcal{H}_2(Y_0, m_0)$ and $X_0 = g^{a_0}$. There is also the requirement that (m_0, Y_0) did not appear in any of $\hat{B}$'s responses to the forger's signature queries. The XCR security definition in [41] incorrectly states that the forger's output (m_0, Y_0, σ) should satisfy $Y_0 \neq 0$ and $\sigma = X_0^{y_0 + \mathrm{E}b}$ where $Y = g^{y_0}$. The latter condition is *not* equivalent to the condition $\sigma = (Y_0 B^{\mathrm{E}})^{a_0}$ in the case where $Y_0 \notin \mathcal{G}$ — indeed y_0 is not even defined when $Y_0 \notin \mathcal{G}$. Now, in order to compute $\mathrm{CDH}(B, X_0)$, the forger is run twice with input B, X_0. The forger's hash function and signature queries are suitably answered so that the two invocations of the forger eventually produce valid forgeries (m_0, Y_0, σ) and (m_0, Y_0, σ') where $\mathrm{E} = \mathcal{H}_2(Y_0, m_0)$, $\mathrm{E}' = \mathcal{H}_2'(Y_0, m_0)$, and $\mathrm{E} \not\equiv \mathrm{E}' \pmod q$. To conclude the argument, one notes that

$$\frac{\sigma}{\sigma'} = \frac{(Y_0 B^{\mathrm{E}})^{x_0}}{(Y_0 B^{\mathrm{E}'})^{x_0}} = (B^{x_0})^{\mathrm{E}-\mathrm{E}'} \tag{6.1}$$

whence $\mathrm{CDH}(B, X_0) = (\sigma/\sigma')^{(\mathrm{E}-\mathrm{E}')^{-1}}$ can be efficiently computed.

The flaw in this argument is the assumption that the Y_0 terms in (6.1) can be cancelled under the sole condition that $Y_0 \neq 0$. While the cancellation in (6.1) is valid if $Y_0 \in \mathcal{G}$ (which is the case if Y_0 has been validated), in general one needs to make additional assumptions including that Y_0 is invertible. Thus, since the description of XCR does not mandate that the verifier validate Y, the XCR security proof in [41] is incorrect.

This flaw in the XCR security proof accounts for the following attack on XCR. Let R and Z be as defined in §6.7.1, and suppose for the sake of concreteness that R is commutative. A forger can respond to $\hat{A}$'s challenge (X, m) with the signature $(Y = Z, \sigma = 0)$. The signature is accepted by $\hat{A}$ since $Z \neq 0$ and $(ZB^{\mathrm{E}})^x = 0$. This attack on XCR in turn explains why the attack described in §6.7.1 can be launched on HMQV.

6.8 No static public-key validation

Consider the hypothetical situation where ephemeral public keys are validated but static public keys are not validated. As mentioned, this situation is worth investigating because validation for static public keys may be omitted if a certification authority is not configured to perform such tests. The attacks can be mounted in the realistic setting where $\mathcal{G}$ is a DSA-type group or an elliptic curve group.

6.8.1 In DSA-type groups

Suppose that $\mathcal{G}$ is the order-q subgroup of $\mathbb{F}_p^*$, and that $t = 2^r$ is a divisor of $(p-1)/q$. Let $\gamma \in \mathbb{F}_p^*$ be an element of order t. Using the notation introduced in §6.2.1, $R = \mathbb{F}_p$ and $\mathcal{G}' = \langle\gamma\rangle$.

The adversary $\mathcal{M}$, who controls $\hat{M}$, selects a valid $Z \in \mathcal{G}$, computes $\mathrm{D} = \mathcal{H}_2(Z, \hat{B}) \bmod q$ and $M = \gamma Z^{-\mathrm{D}^{-1} \bmod q}$. The adversary certifies M as $\hat{M}$'s (invalid) static public key and sends Z to $\hat{B}$ who computes $\beta = ZM^{\mathrm{D}} = \gamma^{\mathrm{D}}$ and $\kappa = \mathcal{H}(\beta^{s_B})$. As in the attack described in §6.6, $\mathcal{M}$ learns y, κ, and $s_B \bmod t$; repeating this procedure yields half the bits of b. The remaining bits of b can thereafter be determined in $O(q^{1/4})$ time using Pollard's lambda method [63].

6.8.2 In elliptic curve groups

Suppose that $\mathcal{G} = E(\mathbb{F}_p)$ where $E : V^2 = U^3 + \alpha U + \beta$ is an elliptic curve of prime order defined over the prime field $\mathbb{F}_p$. Let $P_1 = (u_1, v_1)$ and $P_2 = (u_2, v_2)$ be two finite points in $E(\mathbb{F}_p)$ with $P_1 \neq -P_2$, and let $P_3 = (u_3, v_3) = P_1 + P_2$. The usual formulae for computing P_3 are:

$$u_3 = \lambda^2 - u_1 - u_2, \tag{6.2}$$

$$v_3 = \lambda(u_1 - u_3) - v_1, \tag{6.3}$$

where

$$\lambda = \frac{v_2 - v_1}{u_2 - u_1} \text{ or } \lambda = \frac{3u_1^2 + \alpha}{2v_1},$$

depending on whether $P_1 \neq P_2$ or $P_1 = P_2$. Note that the formulae do not (explicitly) depend on the coefficient β.

The adversary $\mathcal{M}$'s goal is to select two points $M, Z \in \mathbb{F}_p \times \mathbb{F}_p$ such that (i) Z is valid, i.e., $Z \in E(\mathbb{F}_p)$, $Z \neq \infty$; and (ii) $T = Z + \mathrm{D}M$ is a point of order 16 on some curve $E' : V^2 = U^3 + \alpha U + \beta'$ defined over $\mathbb{F}_p$, where $\mathrm{D} = \mathcal{H}_2(Z, \hat{B})$ and $Z + \mathrm{D}M$

is computed using the formulae for $E(\mathbb{F}_p)$. Using the notation introduced in §6.6, $R = E'(\mathbb{F}_p)$, $\mathcal{G}' = \langle T \rangle$, and $t = 16$. The adversary then certifies M as the (invalid) static public key $\hat{M}$ and sends Z to $\hat{B}$, who computes $\kappa = \mathcal{H}(s_B T)$. As in the attack described in §6.6, $\mathcal{M}$ learns y, κ, and $s_B \bmod t$; repeating this procedure yields half the bits of b. The remaining bits of b can thereafter be determined in $O(q^{1/4})$ time using Pollard's lambda method [63].

The adversary $\mathcal{M}$ can proceed to determine M and Z as follows: $\mathcal{M}$ first selects an arbitrary finite point $Z = (u_2, v_2) \in E(\mathbb{F}_p)$ such that $\mathrm{D} = \mathcal{H}_2(Z, \hat{B})$ is odd. Now let $M = (z, 0)$, where $z \in \mathbb{F}_p$ is an indeterminate whose value will be specified later. Since D is odd, application of the group law for E yields $\mathrm{D}M = M$. The coordinates (u_3, v_3) of $T = Z + \mathrm{D}M$ are then derived using (6.2) and (6.3):

$$u_3 = \left(\frac{v_2}{u_2 - z}\right)^2 - z - u_2 \quad \text{and} \quad v_3 = \frac{v_2}{u_2 - z}(z - u_3). \tag{6.4}$$

Define

$$\beta' = v_3^2 - u_3^3 - \alpha u_3, \tag{6.5}$$

so that $T = (u_3, v_3)$ is an $\mathbb{F}_p$-point on the elliptic curve

$$E' : V^2 = U^3 + \alpha U + \beta'. \tag{6.6}$$

The adversary can use division polynomials to select $z \in \mathbb{F}_p$ so that T has order 16. The following result is well known (e.g., see [67]).

Theorem 6.8.1 *Consider the division polynomials $\Psi_k(U, V) \in \mathbb{F}_p[U, V]$ associated with an elliptic curve $E/\mathbb{F}_p : V^2 = U^3 + \alpha U + \beta$ and defined recursively as follows:*

$$\begin{aligned}
\Psi_1(U,V) &= 1 \\
\Psi_2(U,V) &= 2V \\
\Psi_3(U,V) &= 3U^4 + 6\alpha U^2 + 12\beta U - \alpha^2 \\
\Psi_4(U,V) &= 4V(U^6 + 5\alpha U^4 + 20\beta U^3 - 5\alpha^2 U^2 - 4\alpha\beta U - 8\beta^2 - \alpha^3) \\
\Psi_{2k+1}(U,V) &= \Psi_{k+2}\Psi_k^3 - \Psi_{k+1}^3\Psi_{k-1} \text{ for } k \geq 2 \\
\Psi_{2k}(U,V) &= \Psi_k(\Psi_{k+2}\Psi_{k-1}^2 - \Psi_{k-2}\Psi_{k+1}^2)/2V \text{ for } k \geq 3.
\end{aligned}$$

Let Ψ'_k be the polynomial obtained by repeatedly replacing occurrences of V^2 in Ψ_k by $U^3 + \alpha U + \beta$, and define

$$f_k = \begin{cases} \Psi'_k(U,V), & \text{if } k \text{ is odd}, \\ \Psi'_k(U,V)/V, & \text{if } k \text{ is even}. \end{cases}$$

Then in fact $f_k \in \mathbb{F}_p[U]$. Moreover, if $P = (u, v) \in E(\overline{\mathbb{F}_p})$ such that $2P \neq \infty$, then $kP = \infty$ if and only if $f_k(u) = 0$.

It follows from Theorem 6.8.1 that the roots of the polynomial

$$g(U) = \frac{f_{16}(U)}{f_8(U)}$$

are precisely the U-coordinates of points of order 16 in $E(\overline{\mathbb{F}_p})$, and hence $\deg(g) = 96$.

Now to determine T, the adversary computes $h(z) = g(u_3)$, where $g(U)$ is associated with $E' : V^2 = U^3 + \alpha U + \beta'$, and where u_3 and β' are defined in (6.4) and (6.5). It can be seen that $h(z) = h_1(z)/h_2(z)$, where $h_1, h_2 \in \mathbb{F}_p[z]$ and $\deg(h_1) = 288$. More generally, if $t = 2^r$, then $\deg(g) = 3 \cdot 2^{2r-3}$ and $\deg(h_1) = 9 \cdot 2^{2r-3}$. If the polynomial h_1 has a root z in $\mathbb{F}_p$, then the associated point T is guaranteed to have order 16 in $E'(\mathbb{F}_p)$. Since Z can be chosen uniformly at random from $E(\mathbb{F}_p)$, it is reasonable to make the heuristic assumption that h_1 is a "random" degree-288 polynomial over $\mathbb{F}_p$. The following result ensures that there is a very good chance that h_1 will indeed have a root in $\mathbb{F}_p$. The result is well known (e.g., see [39, §4.6.2, Excercise 1]). The proof is included for the sake of completeness.

Theorem 6.8.2 *For $p \gg n \geq 10$, the proportion of degree-n polynomials over $\mathbb{F}_p$ that have at least one root in $\mathbb{F}_p$ is approximately $(1 - \frac{1}{e}) \approx 0.632$.*

Proof: It suffices to consider monic polynomials over $\mathbb{F}_p$.

The generating function for the number of monic polynomials over $\mathbb{F}_p$ with respect to degree is

$$\Phi(x) = \sum_{i \geq 0} p^i x^i = \frac{1}{1 - px}. \tag{6.7}$$

Let $L(n, p)$ denote the number of degree-n monic irreducible polynomials over $\mathbb{F}_p$. Since every monic polynomial can be written as a product of monic irreducible polynomials, the generating function $\Phi(x)$ can be written as

$$\Phi(x) = \prod_{i \geq 1} \left(\frac{1}{1 - x^i} \right)^{L(i,p)}. \tag{6.8}$$

Now, the generating function for monic polynomials with no linear factors (i.e., no roots in $\mathbb{F}_p$) is

$$\tilde{\Phi}(x) = \prod_{i \geq 2} \left(\frac{1}{1 - x^i} \right)^{L(i,p)}. \tag{6.9}$$

Multiplying (6.8) by $(1 - x)^{L(1,p)} = (1 - x)^p$ yields

$$\tilde{\Phi}(x) = \frac{(1 - x)^p}{1 - px}. \tag{6.10}$$

Letting $[\cdot]$ denote the coefficient operator, it follows from (6.10) that the number $R(n,p)$ of monic polynomials of degree n over $\mathbb{F}_p$ that have at least one root in $\mathbb{F}_p$ is

$$R(n,p) = p^n - [x^n]\tilde{\Phi}(x) = p^n - \sum_{i=0}^{n} \binom{p}{i}(-1)^i p^{n-i}.$$

Therefore, for $p \gg n \geq 10$

$$R(n,p) \approx p^n \sum_{i=1}^{n} \frac{(-1)^{i-1}}{i!} \approx p^n \sum_{i\geq 1} \frac{(-1)^{i-1}}{i!} = p^n \left(1 - \frac{1}{e}\right),$$

which completes the argument. □

Example. Determination of M, Z, T and E': Consider the NIST-recommended elliptic curve [29] defined by the equation $E : V^2 = U^3 - 3U + \beta$ over $\mathbb{F}_p$, where $p = 2^{192} - 2^{64} - 1$ and

$$\beta = 245515554600894381774029391519745178476910805816119123806 5.$$

Suppose that $\mathcal{M}$ selects Z

$$\begin{aligned} Z \;=\; & (6020462823756886567582134805875261119166989766368846848 18, \\ & 1740503322936220314048575522802194103640234889273866506 41) \end{aligned}$$

in $E(\mathbb{F}_p)$, and

$$M = (266459051458792235985361256551627093778386698181279825085 1,\ 0).$$

Then the point $T = Z + M$ computed using the group law for $E(\mathbb{F}_p)$ is

$$\begin{aligned} T \;=\; & (5350077178842604929587851454217201721791103389533004256989, \\ & 4170329249603673452251890924513609385018269372344921771517). \end{aligned}$$

T is a point of order 16 on $E' : y^2 = x^3 - 3x + \beta'$, where

$$\beta' = 227183583666963229242395349868046014316554092275124653862 7.$$

6.9 No ephemeral public-key validation

Consider the hypothetical situation where static public keys are validated but ephemeral public keys are not validated. No attacks on HMQV are known in the case where the underlying group $\mathcal{G}$ is a DSA-type group or an elliptic curve group.

In particular, it is not clear how to extend the attacks described in §6.8.1 and §6.8.2 to this setting. The difficulty is in part because of the complicated relationship between A and $\mathrm{D} = \mathcal{H}_2(X, \hat{B})$ whereby D is not determined until X has been fixed.

However, the attacks similar to ones described in §6.8.1 and §6.8.2 can be launched on MQV if ephemeral public keys are not validated. Suppose that $\mathcal{G} = E(\mathbb{F}_p)$ where $E/\mathbb{F}_p : V^2 = U^3 + \alpha U + \beta$ is an elliptic curve of prime order. The adversary $\mathcal{M}$, who wishes to impersonate $\hat{A}$ to $\hat{B}$, selects $u_1 \in_R \mathbb{F}_p$ and sets $Z = (u_1, z)$ where z is an indeterminate. Since in MQV the exponent D depends only on u_1, $\mathcal{M}$ can then compute $M = \mathrm{D}A$, where A is $\hat{A}$'s (valid) static public key. Using the method of §6.8.2, $\mathcal{M}$ can use the t-th division polynomial (for some small t) to determine $z, \beta' \in \mathbb{F}_p$ so that $T = Z + M$ has order t on $E' : V^2 = U^3 + \alpha U + \beta'$. The adversary sends Z to $\hat{B}$ who computes the session key $\kappa = \mathcal{H}(s_B T, \hat{A}, \hat{B})$. Now $\mathcal{M}$ can guess the session key with non-negligible success probability $\frac{1}{t}$. Alternatively, if $\mathcal{M}$ can learn $\hat{B}$'s ephemeral private key y, then $\mathcal{M}$ can determine $\hat{B}$'s static private key b as in §6.6.

6.10 Partial validation

It may be possible to circumvent the attacks described in the preceding sections without performing (full) public-key validation on static and ephemeral public keys. For example, consider the version of HMQV that has recently been proposed for standardization by the IEEE 1363 standards group [42]. This proposal specifies HMQV in the concrete setting of a DSA-type group $\mathcal{G}$. The only checks required on ephemeral and static public keys is that they be integers in the interval $[2, p-1]$. In [42] it is claimed that this instantiation of HMQV is provably secure (under the assumptions that CDH in $\mathcal{G}$ is intractable, and that the employed hash functions are random functions) as long as ephemeral private keys are never leaked. Moreover, in order to resist attacks that may be mounted in the face of ephemeral private key leakage, the recipient of an ephemeral key X and static key A only needs to verify that $T^q = 1$ and $T \neq 1$ where $T = XA^{\mathrm{D}}$. Such a check is more efficient that separately verifying that $A^q = 1$ and $X^q = 1$. Again, [42] claims that this version of HMQV is provably secure even if the adversary is able to learn some ephemeral private keys.

6.11 Almost validation

A public key X is said to have been *almost validated* if it has been verified that $X \in \mathcal{G}$ but not necessarily that $X \neq 1$. Protocol descriptions sometimes inadvertently omit the condition $X \neq 1$ (see '$\mathcal{G}$-tests' in [41]). Performing almost validation instead of full validation of public keys may lead to new vulnerabilities. An example of this likelihood follows.

In the one-pass HMQV protocol [41], only the initiator contributes an ephemeral public key. The initiator $\hat{A}$ sends $(\hat{A}, \hat{B}, X)$ to $\hat{B}$ and computes the session key $\kappa = \mathcal{H}(B^{s_A})$ where $s_A = x + \mathrm{D}a \bmod q$ and $\mathrm{D} = \mathcal{H}_2(X, \hat{A}, \hat{B})$. The receiver $\hat{B}$ verifies that $X \neq 0$ and computes $\kappa = \mathcal{H}((XA^{\mathrm{D}})^b)$.

In [52] Menezes showed that the one-pass HMQV protocol succumbs to a Kaliski-style unknown-key share attack [18] even if public keys are (fully) validated. The attack is 'on-line' in the sense that the adversary needs to have her static public key certified during the attack. The next attack is an 'off-line' Kaliski-style attack on the one-pass HMQV protocol which succeeds if ephemeral public keys are (fully) validated but static public keys are only almost validated.

The adversary $\mathcal{M}$, who controls $\hat{M}$, registers in advance the static public key $M = 1$ with the certification authority. Now, when $\hat{A}$ sends $(\hat{A}, \hat{B}, X)$, $\mathcal{M}$ replaces this message with $(\hat{A}, \hat{M}, Z)$ where $Z = XA^{\mathrm{D}}$ and $\mathrm{D} = \mathcal{H}_2(X, \hat{A}, \hat{B})$. Note that Z is valid, whereas M is only partially valid. The recipient $\hat{B}$ computes $\mathrm{D}' = \mathcal{H}_2(Z, \hat{M}, \hat{B})$ and

$$\kappa = \mathcal{H}((ZM^{\mathrm{D}'})^b) = \mathcal{H}(Z^b) = \mathcal{H}((XA^{\mathrm{D}})^b).$$

Thus $\hat{A}$ and $\hat{B}$ have computed the same session key, but $\hat{B}$ mistakenly believes that the key is shared with $\hat{M}$.

6.12 Validation summary

The attacks on HMQV presented in §6.6, §6.7.1 and §6.8 are also effective on MQV if validation of static or ephemeral public keys is omitted. The attacks are summarized in Table 6.1. While these attacks are not necessarily practical and may not be a threat in real-world settings, they nonetheless illustrate the importance of performing some form of validation for static and ephemeral public keys in Diffie-Hellman type key agreement protocols. Furthermore, the attacks highlight the danger of relying on security proofs for discrete-logarithm protocols where a concrete representation for the underlying group is not specified. In particular, since public keys

in HMQV are not necessarily valid, the security of HMQV depends on several aspects of the representation for the underlying group $\mathcal{G}$ including the manner in which the group operation is performed, and the particular algorithm chosen for computing $(XA^{\mathrm{D}})^{s_B}$ and $(YB^{\mathrm{E}})^{s_A}$. For other examples of the pitfalls when relying on security proofs where a concrete representation of the underlying group is not specified, see [57] and [70].

Static public keys validated?	Ephemeral public keys validated?	Ephemeral private keys secure?	Attacks
√	√	√	No attack known
√	√	×	No attack known
×	√	√	No attack known
√	×	√	§6.7.1, §6.9†
√	×	×	§6.7.1, §6.9†
×	×	√	§6.7.1, §6.9†
×	√	×	§6.8.1, §6.8.2
×	×	×	§6.6, §6.7.1, §6.8.1, §6.8.2, §6.9†

Table 6.1: Summary of attacks on HMQV (and MQV without validation).
† The attack of §6.9 applies to MQV only.

6.13 Concluding remarks

Even though efficiency is highly demanded, the examples presented here show that employing speedups without careful considerations may lead to security breaches. A natural question that arises is to look for efficient validation techniques and develop protocols that allow for reusing ephemeral key pairs.

Chapter 7

Conclusions and future work

The main research topic of this thesis was key establishment schemes focusing on the security models and assurances provided by security definitions. Different models and protocols were presented, compared and analysed. The following is an outline of questions and issues addressed in the previous chapters:

- Relationship between models and security attributes. Given a set of security attributes, does a model encompass that set of attributes? Given a security model, what security guarantees are provided?
- Model enhancement. There are known examples of security attributes that are not captured by existing definitions. New models were developed that can account for a wider range of possible attacks.
- Protocol design. The ability to combine previous knowledge to develop new efficient protocols that meet a wider range of security goals is essential for further development of key agreement.
- Security arguments. Both creating new arguments and critically reading existing arguments are important for the acceptance and understanding of protocols.

Chapter 6 presented an issue related to key establishment schemes that is not addressed in the models. In particular efficiency considerations are an important aspect in the deployment of a given key establishment scheme. Reusing ephemeral public keys appears to be a simple way to increase the efficiency of key agreement protocols. Naturally, the next step is to design protocols that allow reusing the same ephemeral key. The static key pair of a party is reused in multiple sessions. Hence

it should be possible to reuse ephemeral keys in multiple sessions without affecting security attributes. However, if key independence is important, then the protocol description should provide means for introducing session specific freshness.

It is common to have a multitude of key establishment schemes to choose from — for example SP800-56A [71] allows parties to choose either the MQV or the UM protocol. It is generally accepted that using the same private key for more than one protocol is not sound cryptographic practice. Hence a party that wishes to be fully SP800-56A compliant, which does not prevent static public keys to be used in different protocols, and at the same time follow sound cryptographic practices should have two different static key pairs — one for use with MQV, and one for use with UM. However, if the underlying groups are the same, for simplicity a party may establish only one static key pair. A natural question to ask is if the same static key can be used with two different protocols and formally argue the soundness of this use. Key agreement models are independent from protocol descriptions, and therefore the existing models could potentially be used to validate that static private information used in one protocol can be safely used in another protocol. Furthermore, if the two protocols provide different assurances (e.g., one protocol is KCI resilient and the other is not), what are the implications for the party? In particular does the KCI resilient protocol achieve less if the static public key is used in a protocol that does not provide KCI resilience.

The above idea of sharing static key pairs among different protocols decreases the amount of work a party must invest in static key pair management — both in terms of number of the keys the party owns and in terms of the number of certificates the party receives from its peers. If on-line efficiency is an issue, then just like sharing static key pairs among different protocols, sharing ephemeral key pairs among different protocols should be considered. Potentially this could have a great impact on efficiency, hence it is an issue worth considering. This type of work can further be considered for reusing ephemeral key pairs among different protocols and among different sessions, improving performance even further.

It is plausible to assume that with the correct set of cryptographic assumptions, the above questions have affirmative answers. Since the protocols considered here were Diffie-Hellman type key agreement protocols, it is interesting to pursue the above questions based on the CDH or the GDH assumptions, possibly using the random oracle assumption.

Denial-of-Services (DoS) is an important practical aspect that is related to efficiency. Countermeasures against DoS attacks are incorporated on top of existing key agreement protocols. The analysed key agreement models and definitions do

not take into account DoS attacks. Furthermore, the design principles of key agreement protocols do not attempt to make use of the DoS measures that are employed by the parties. However, it may be possible to combine the key agreement protocols with DoS countermeasures to improve existing protocols. Until now these issues were considered separately — largely due to the fact that defining DoS formally is a non-trivial task — but combining the analysis of protocols and DoS countermeasures could lead to simpler protocols, security arguments and improved DoS resilience.

The main assurances of the existing models are geared towards security. But in many cases there are alternative requirements that are no less important than security. For example the CK02 model and the combined model in §5.4 appear to open venues for anonymity considerations. However, in the models the adversary controls all communication. Hence the adversary is aware where messages originate and where messages are delivered. It is not immediately clear how anonymity can be added.

Much of the focus until now was on security of key agreement protocols. But how is a session key supposed to be used in subsequent communications? A key agreement protocol provides certain assurances but can those assurances contradict the intended use of the key agreement protocol? Different cryptographic primitives have different security definitions. If a session key is going to be an input to a cryptographic primitive how are the security notions related? Should the key agreement and the subsequent protocol be considered at once or there is a way to establish that the definitions are independent from each other? An even more interesting question is if the session key already guarantees a certain security property, that the next protocol Π requires, is it possible to simplify Π's security definition. This is a desirable approach since it could simplify the definitions and hence increase acceptance of new cryptographic algorithms, as their analysis will be more elementary. Some of these questions have been addressed by the research on "Universal Composability"; for example see [20], [21] and [19].

Bibliography

[1] William Aiello, Steven M. Bellovin, Matt Blaze, Ran Canetti, John Ioannidis, Angelos D. Keromytis, and Omer Reingold, *Just fast keying: Key agreement in a hostile Internet*, ACM Transactions on Information and System Security **7** (2004), no. 2, 242–273.

[2] ANSI X9.42, *Public key cryptography for the financial services industry: Agreement of symmetric keys using discrete logarithm cryptography*, American National Standards Institute, 2003.

[3] ANSI X9.63, *Public key cryptography for the financial services industry: Key agreement and key transport using elliptic curve cryptography*, American National Standards Institute, 2001.

[4] Adrian Antipa, Daniel Brown, Alfred Menezes, René Struik, and Scott Vanstone, *Validation of elliptic curve public keys*, Public Key Cryptography – PKC 2003 (Miami, FL, USA), Lecture Notes of Computer Science, vol. 2567, Springer Verlag, 2003, pp. 211–223.

[5] Endre Bangerter, Jan Camenisch, and Ueli Maurer, *Efficient proofs of knowledge of discrete logarithms and representations in groups with hidden order*, Public Key Cryptography – PKC 2005 (Les Diablerets, Switzerland), Lecture Notes of Computer Science, vol. 3386, Springer Verlag, 2005, pp. 154–171.

[6] William D. Banks and Igor Shparlinski, *Integers with a large smooth divisor*, Integers: Electronic Journal of Combinatorial Number Theory **7** (2007), no. A17, 1–11.

[7] Mihir Bellare and Adriana Palacio, *The knowledge-of-exponent assumptions and 3-round zero-knowledge protocols*, Advances in Cryptology – CRYPTO 2004 (Santa Barbara, CA, USA), Lecture Notes of Computer Science, vol. 3152, Springer Verlag, 2004, pp. 273–289.

[8] Mihir Bellare, David Pointcheval, and Phillip Rogaway, *Authenticated key exchange secure against dictionary attacks*, Advances in Cryptology – EUROCRYPT 2000 (Bruges, Belgium), Lecture Notes of Computer Science, vol. 1807, Springer Verlag, 2000, pp. 139–155.

[9] Mihir Bellare and Phillip Rogaway, *Entity authentication and key distribution*, Advances in Cryptology – CRYPTO'93 (Santa Barbara, CA, USA), Lecture Notes of Computer Science, vol. 773, Springer Verlag, 1993, Full version available at http://www.cs.ucdavis.edu/~rogaway/papers/eakd-abstract.html, pp. 232–249.

[10] _____, *Random oracles are practical: a paradigm for designing efficient protocols*, CCS'93: Proceedings of the 1st ACM Conference on Computer and Communications Security (New York, NY, USA), ACM, 1993, pp. 62–73.

[11] _____, *Provably secure session key distribution: the three party case*, STOC'95: Proceedings of the 27th annual ACM Symposium on Theory of Computing (Las Vegas, NE, USA), ACM, 1995, pp. 57–66.

[12] Ingrid Biehl, Bernd Meyer, and Volker Müller, *Differential fault attacks on elliptic curve cryptosystems*, Advances in Cryptology – CRYPTO 2000 (Santa Barbara, CA, USA), Lecture Notes of Computer Science, vol. 1880, Springer Verlag, 2000, pp. 131–146.

[13] Simon Blake-Wilson, Don Johnson, and Alfred Menezes, *Key agreement protocols and their security analysis*, Cryptography and Coding: 6th IMA International Conference (Cirencester, UK), Lecture Notes of Computer Science, vol. 1355, Springer Verlag, 1997, pp. 30–45.

[14] Simon Blake-Wilson and Alfred Menezes, *Unknown key-share attacks on the station-to-station (STS) protocol*, Public Key Cryptography – PKC'99 (Kamakura, Japan), Lecture Notes of Computer Science, vol. 1560, Springer Verlag, 1999, pp. 154–170.

[15] Colin Boyd, Wenbo Mao, and Kenneth G. Paterson, *Key agreement using statically keyed authenticators*, in Jakobsson et al. [36], pp. 248–262.

[16] Colin Boyd and Anish Mathuria, *Protocols for authentication and key establishment*, Springer Verlag, Germany, 2003.

[17] Daniel Brown, *Additional ECC groups for IKE and IKEv2*, Internet Engineering Task Force, October 2006, Internet draft.

[18] Jr. Burton S. Kaliski, *An unknown key-share attack on the MQV key agreement protocol*, ACM Transaction on Information and System Security **4** (2001), no. 3, 275–288.

[19] Ran Canetti, *Universally composable security: A new paradigm for cryptographic protocols*, 42nd IEEE symposium on Foundations of Computer Science (FOCS'01) (Los Alamitos, CA, USA), IEEE Computer Society, 2001, pp. 136–147.

[20] Ran Canetti and Hugo Krawczyk, *Analysis of key-exchange protocols and their use for building secure channels*, Advances in Cryptology – EUROCRYPT 2001 (Aarhus, Denmark), Lecture Notes of Computer Science, vol. 2045, Springer Verlag, 2001, Full version available at http://eprint.iacr.org/2001/040/, pp. 453–474.

[21] ———, *Security analysis of IKE's signature-based key-exchange protocol*, Advances in Cryptology – CRYPTO 2002 (Moti Yung, ed.), Lecture Notes of Computer Science, vol. 2442, Springer, 2002, Full version available at http://eprint.iacr.org/2002/120/, pp. 143–161.

[22] Liqun Chen, Zhaohui Cheng, and Nigel P. Smart, *Identity-based key agreement protocols from pairings*, International Journal of Information Security **6** (2007), no. 4, 213–241.

[23] Kim-Kwang Raymond Choo, Colin Boyd, and Yvonne Hitchcock, *Examining indistinguishability-based proof models for key establishment protocols*, Advances in Cryptology – ASIACRYPT 2005 (Chennai, India), Lecture Notes of Computer Science, vol. 3788, Springer Verlag, 2005, pp. 585–604.

[24] Whitfield Diffie and Martin E. Hellman, *New directions in cryptography*, IEEE Transactions on Information Theory **IT-22** (1976), no. 6, 644–654.

[25] Whitfield Diffie, Paul C. van Oorschot, and Michael J. Wiener, *Authentication and authenticated key exchange*, Designs, Codes and Cryptography **2** (1992), no. 2, 107–125.

[26] Blake Ramsdell (editor), *Secure/multipurpose internet mail extensions (S/MIME) version 3.1 message specification*, Internet Engineering Task Force, 2004, RFC 3851.

[27] Charlie Kaufman (editor), *Internet key exchange (IKEv2) protocol*, Internet Engineering Task Force, 2005, RFC 4306.

[28] Tahir ElGamal, *A public key cryptosystem and a signature scheme based on discrete logarithms*, IEEE Transactions on Information Theory **IT-31** (1985), no. 4, 469–472.

[29] FIPS186-2, *Federal information processing standards publication 186-2, digital signature standard (DSS)*, National Institute of Standards and Technology, January 2000.

[30] David Fu and Jerry Solinas, *ECP groups for IKE and IKEv2*, Internet Engineering Task Force, January 2007, RFC 4753.

[31] Dan Harkins and Dave Carrel, *The Internet key exchange (IKE)*, Internet Engineering Task Force, 1998, RFC 2409.

[32] Russell Housley, *Triple-DES and RC2 key wrapping*, Internet Engineering Task Force, 2001, RFC 3217.

[33] ———, *Cryptographic message syntax (CMS) algorithms*, Internet Engineering Task Force, 2002, RFC 3370.

[34] ———, *Cryptographic message syntax (CMS)*, Internet Engineering Task Force, 2004, RFC 3852.

[35] IEEE Std 1363-2000, *IEEE standard specifications for public-key cryptography*, 2000.

[36] Markus Jakobsson, Moti Yung, and Jianying Zhou (eds.), *Applied cryptography and network security, second international conference, ACNS 2004, Proceedings*, Lecture Notes of Computer Science, vol. 3089, Yellow Mountain, China, Springer Verlag, 2004.

[37] Ik Rae Jeong, Jonathan Katz, and Dong Hoon Lee, *One-round protocols for two-party authenticated key exchange*, in Jakobsson et al. [36], pp. 220–232.

[38] Mike Just and Serge Vaudenay, *Authenticated multi-party key agreement*, Advances in Cryptology – ASIACRYPT'96 (Kyongju, Korea), Lecture Notes of Computer Science, vol. 1163, Springer Verlag, 1996, pp. 36–49.

[39] Donald E. Knuth, *Seminumerical algorithms*, 3 ed., vol. 2, Addison-Wesley, Reading, MA, USA, 1997.

[40] Hugo Krawczyk, *SIGMA: The 'SIGn-and-MAc' approach to authenticated Diffie-Hellman and its use in the IKE-protocols*, Advances in Cryptology – CRYPTO 2003 (Dan Boneh, ed.), Lecture Notes of Computer Science, vol. 2729, Springer, 2003, pp. 400–425.

[41] ———, *HMQV: A high-performance secure Diffie-Hellman protocol*, Advances in Cryptology – CRYPTO 2005 (Santa Barbara, CA, USA), Lecture Notes of Computer Science, vol. 3621, Springer Verlag, 2005, Full version available at http://eprint.iacr.org/2005/176, pp. 546–566.

[42] ———, *HMQV in IEEE P1263*, July 2006, submission to the IEEE P1263 working group http://grouper.ieee.org/groups/1363/P1363-Reaffirm/submissions/krawczyk-hmqv-spec.pdf.

[43] Sébastien Kunz-Jacques, Gwenaëlle Martinet, Guillaume Poupard, and Jacques Stern, *Cryptanalysis of an efficient proof of knowledge of discrete logarithm*, in Yung [74], pp. 27–43.

[44] Sébastien Kunz-Jacques and David Pointcheval, *About the security of MTI/C0 and MQV*, Security and Cryptography for Networks – SCN 2006 (Maiori, Italy), Lecture Notes of Computer Science, vol. 4116, Springer Verlag, 2006, pp. 156–172.

[45] Brian LaMacchia, Kristin Lauter, and Anton Mityagin, *Stronger security of authenticated key exchange*, Provable Security: First International Conference, ProvSec 2007 (Wollongong, Australia), Lecture Notes of Computer Science, vol. 4784, Springer Verlag, 2007, pp. 1–16.

[46] Kristin Lauter and Anton Mityagin, *Security analysis of KEA authenticated key exchange protocol*, in Yung [74], pp. 378–394.

[47] Laurie Law, Alfred Menezes, Minghua Qu, Jerry Solinas, and Scott Vanstone, *An efficient protocol for authenticated key agreement*, Designs, Codes and Cryptography **28** (2003), no. 2, 119–134.

[48] Peter J. Leadbitter and Nigel P. Smart, *Analysis of the insecurity of ECMQV with partially known nonces*, International Security Conference – ISC 2003 (Bristol, UK), Lecture Notes of Computer Science, vol. 2851, Springer Verlag, 2003, pp. 240–251.

[49] Chae Hoon Lim and Pil Joong Lee, *A key recovery attack on discrete log-based schemes using a prime order subgroup*, Advances in Cryptology – CRYPTO'97 (Santa Barbara, CA, USA), Lecture Notes of Computer Science, vol. 1294, Springer Verlag, 1997, pp. 249–263.

[50] Tsutomu Matsumoto, Youichi Takashima, and Hideki Imai, *On seeking smart public-key-distribution systems*, Transactions of the IEICE of Japan **E69** (1986), no. 2, 99–106.

[51] Ueli Maurer and Stefan Wolf, *Diffie-Hellman oracles*, Advances in Cryptology – CRYPTO'96 (Santa Barbara, CA, USA), Lecture Notes of Computer Science, vol. 1109, Springer Verlag, 1996, pp. 268–282.

[52] Alfred Menezes, *Another look at HMQV*, Journal of Mathematical Cryptology **1** (2007), no. 1, 47–64.

[53] Alfred Menezes and Berkant Ustaoglu, *On the importance of public-key validation in the MQV and HMQV key agreement protocols*, Progress in Cryptology – INDOCRYPT 2006 (Kolkata, India), Lecture Notes of Computer Science, vol. 4329, Springer Verlag, 2006, pp. 133–147.

[54] ———, *Comparing the pre- and post-specified peer models for key agreement*, Information Security and Privacy – ACISP 2008 (Yi Mu, Willy Susilo, and Jennifer Seberry, eds.), Lecture Notes of Computer Science, vol. 5107, Springer, 2008, pp. 53–68.

[55] Alfred Menezes, Paul C. van Oorschot, and Scott A. Vanstone, *Handbook of applied cryptography*, CRC Press, Boca Raton, FL, USA, 1997.

[56] Alfred Menezes and Yi-Hong Wu, *The discrete logarithm problem in $GL(n, q)$*, Ars Combinatoria **47** (1998), 23–32.

[57] David Naccache, Nigel P. Smart, and Jacques Stern, *Projective coordinates leak*, Advances in Cryptology – EUROCRYPT 2004 (Interlaken, Switzerland), Lecture Notes of Computer Science, vol. 3027, Springer Verlag, 2004, pp. 257–267.

[58] NIST, *SKIPJACK and KEA algorithm specifications*, National Institute of Standards and Technology, May 1998, Available via http://csrc.nist.gov/groups/STM/documents/skipjack/skipjack.pdf.

[59] Tatsuaki Okamoto, *Authenticated key exchange and key encapsulation in the standard model*, Advances in Cryptology – ASIACRYPT 2007 (Kaoru Kurosawa, ed.), Lecture Notes of Computer Science, vol. 4833, Springer, 2007, pp. 474–484.

[60] Tatsuaki Okamoto and David Pointcheval, *The gap-problems: a new class of problems for the security of cryptographic schemes*, Public Key Cryptography – PKC 2001 (Cheju Island, Korea), Lecture Notes of Computer Science, vol. 1992, Springer Verlag, 2001, pp. 104–118.

[61] David Pointcheval and Jacques Stern, *Security proofs for signature schemes*, Advances in Cryptology – EUROCRYPT'96 (Saragossa, Spain) (Ueli Maurer, ed.), LNCS, vol. 1070, Springer Verlag, 1996, pp. 387–398.

[62] ———, *Security arguments for digital signatures and blind signatures*, Journal of Cryptology **13** (2000), no. 3, 361–396.

[63] John M. Pollard, *Monte Carlo methods for index computation* $\mod p$, Mathematics of Computation **32** (1978), no. 143, 918–924.

[64] Eric Rescorla, *Diffie-Hellman key agreement method*, Internet Engineering Task Force, 1999, RFC 2631.

[65] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman, *A method for obtaining digital signatures and public-key cryptosystems*, Communications of the ACM **21** (1978), no. 2, 120–126.

[66] Jeffrey I. Schiller, *Cryptographic algorithms for use in the internet key exchange version 2 (IKEv2)*, Internet Engineering Task Force, 2005, RFC 4307.

[67] René Schoof, *Elliptic curves over finite fields and the computation of square roots* $\mod p$, Mathematics of Computation **44** (1985), no. 170, 483–494.

[68] Claude E. Shannon, *The mathematical theory of communication*, The Bell System Technical Journal **27** (1938), 379–423, 623–656.

[69] Victor Shoup, *On formal models for secure key exchange*, Available at http://www.shoup.net/papers/, 1999.

[70] Nigel P. Smart, *The exact security of ECIES in the generic group model*, Cryptography and Coding: 8th IMA International Conference (Cirencester, UK), Lecture Notes of Computer Science, vol. 2260, Springer Verlag, 2001, pp. 73–84.

[71] SP 800-56A, *Special publication 800-56A, Recommendation for pair-wise key establishment schemes using discrete logarithm cryptography*, National Institute of Standards and Technology, March 2006.

[72] Douglas R. Stinson, *Cryptography theory and practice*, CRC Press, Boca Raton, FL, USA, 1995.

[73] Berkant Ustaoglu, *Obtaining a secure and efficient key agreement protocol from (H)MQV and NAXOS*, Designs, Codes and Cryptography **46** (2008), no. 3, 329–342.

[74] Moti Yung (ed.), *9th international conference on theory and practice in public-key cryptography, PKC 2004, Proceedings*, Lecture Notes of Computer Science, vol. 3958, New York, NY, USA, Springer Verlag, 2006.

[75] Robert Zuccherato, *Methods for avoiding the "small-subgroup" attacks on the Diffie-Hellman key agreement method for S/MIME*, Internet Engineering Task Force, 2000, RFC 2785.

List of notation

$\mathcal{G}$	multiplicatively written group
A,B,M,U,V,X,Y, T,Z	group elements
$\mathcal{G}^*$	non-identity elements in a group
g	generator of $\mathcal{G}$
q	group order, typically a prime number
$\mathbb{F}_p$	finite field of order p
$\mathbb{Z}_p$	integers modulo p
CDH	Computational Diffie-Hellman problem
GDH	Gap Diffie-Hellman problem
DDH	Decision Diffie-Hellman problem
CSP	Computational Square Diffie-Hellman problem
$\hat{A}$,$\hat{B}$,$\hat{C}$,$\hat{D}$	honest parties
X	$\hat{A}$'s ephemeral public key
x,$\tilde{x}$	$\hat{A}$'s ephemeral private key
A	$\hat{A}$'s static public key
a	$\hat{A}$'s static private key
N_A	$\hat{A}$'s nonce
T_A	$\hat{A}$'s tag
Y	$\hat{B}$'s ephemeral public key
y,$\tilde{y}$	$\hat{B}$'s ephemeral private key
B	$\hat{B}$'s static public key
b	$\hat{B}$'s static private key
N_B	$\hat{B}$'s nonce
T_B	$\hat{B}$'s tag
$\mathcal{I}$	initiator
$\mathcal{R}$	responder
$role$	party's role — responder or initiator
$\mathcal{H}$, $\mathcal{H}_1$,$\mathcal{H}_2$	hash functions, random oracles

$\mathcal{M}$	adversary
$\hat{M}$	adversary controlled party
Z	$\hat{M}$'s ephemeral public key
z	$\hat{M}$'s ephemeral private key
M	$\hat{M}$'s static public key
m	$\hat{M}$'s static private key
λ	global security parameter
$\mathcal{S}$	solver
$\mathbf{n}$	bound on honest parties in a system
$\mathbf{s}$	bound on sessions activated within an honest party
S	session identifier
D,E	public exponents
σ	session shared secret
κ	session secret key