Karina Maria Vallejos

Diffie-Hellman based key exchange

Master's thesis in MLREAL
Supervisor: Kristian Gjøsteen
December 2019

Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Mathematical Sciences

NTNU
Norwegian University of
Science and Technology

Karina Maria Vallejos

Diffie-Hellman based key exchange

Master's thesis in MLREAL
Supervisor: Kristian Gjøsteen
December 2019

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Mathematical Sciences

Acknowledgements

This thesis ends my five year long education at NTNU. This project has been both interesting and educational, and I'm forever grateful for all the memories and friendships I've obtained during my time at NTNU.

On the very outset of this thesis, I would like to extend my sincere and heartfelt gratitude to my supervisor professor Kristian Gjøsteen for his insight and guidance in these long and at times tough months of work before submission. I would like to thank my sister Pamela for all the help I've received during my five years of education. We've spent countless of late nights while studying for my exams and proofread my thesis. I will be forever grateful for the motivation and faith given by my loving and supportive parents, thank you for always giving me guidance and love through thick and thin. I would also like to thank my fiancè, Mathias, who has always supported me morally and always kept my spirits up.

Karina Maria Vallejos
Trondheim, December 2019

Abstract

We present a formalism for the analysis of key exchange protocols that combines different definitional approaches and results - to define security and other important analytical benefits. The use of the results are applied to explain and analyse authenticated key exchange protocols with the use of the Bellare and Rogaway model and standard Diffie-Hellman communication. The definition allows for a simple proof of security: one can design and prove security of key exchange protocols where the communication are authenticated and secure against an active adversary.

Sammendrag

Vi presenterer formaliserte sikkerhetsbevis av flere nøkkelutvekslingsprotokoller som kombinerer ulike definisjonelle tilnærminger og resultater for å definere sikkerhet og andre analytiske fordeler for nøkkelutveksling. Resultatene anvendes til å forklare og analysere autentiserte nøkkelutvekslingsprotokoller ved bruk av Bellare og Rogaway modellen og standard Diffie-Hellman kommunikasjon. Definisjonen fører med seg et enkelt bevis for sikkerhet: man kan designe og bevise sikkerhet for nøkkelutvekslingsprotkoller hvor kommunikasjonen er autentisert og sikker mot en aktiv angriper.

Contents

Chapter 1

Introduction

In this master thesis, our main goal is to explore and analyse the security in key exchange protocols, and introduce additions and improvements to obtain authentication. Key exchange, or KEX, is a traditional primitive of cryptography. The Diffie-Hellman key exchange is a specific method of exchanging cryptographic keys k. These protocols allow two parties to establish and compute a secret key through public communication, over an insecure communication channel. Authenticated key exchange, or AKE, doesn't only allow parties to compute a share key but also establish authenticity of the parties. This ultimately means that a party can compute a shared key only if the player is the one it claims to be. Many of the desirable properties for AKE protocols have been identified, however in this thesis we have mainly considered the notion of *Forward security*. This notion results in the security of established session keys even if the static keys of one or two parties are compromised. A natural solution for AKE is to execute a standard Diffie-Hellman key exchange protocol and sign all communication sent between the parties, which is also referred to as *Signed Diffie-Hellman*. However, the Signed Diffie-Hellman AKE can be broken if an adversary reveals the ephemeral keys of the parties. In addition we wish to decrease the number of exponentiations computed in this protocol and avoid the use of signatures to maintain authentication. Hence we proceed to introduce the *KEA+* protocol and *HMQV* to preserve authentication and the notion of forward security.

The notion of *mutual authentication* and *matching conversation* becomes an important factor in our formalisation of authentication. The terms are introduced by Bellare and Rogaway [1] and easily explained as the case when the conversation of a responder oracle

matches the conversation of an initiator oracle and vice versa.

The thesis will be organized as follows: in Chapter 2, we provide an introduction and overview on the Diffie-Hellman key exchange as well as the cryptographic and mathematical theory around the discrete logarithm security proof; in Chapter 3, we look at authenticated key exchange and introduce a security proof with three games using standard Diffie-Hellman communication and the Bellare-Rogaway model; in Chapter 4, we look at KEA+ in hope to improve previous signed protocol; in Chapter 5; we look further into the notion of *forward security* in our model as well an an extension to our model with the use of the HMQV protocol; in Chapter 6, we summarize our work with some concluding comments.

Chapter 2

Key exchange

In cryptography, key exchange protocols are mechanisms by which keys are exchanged between two parties to allow for the use of a cryptographic algorithm or protocol. The players eventually output either a shared key or $\perp$ that denotes failure. Key exchange protocols (KEX, for short) are essential for allowing the use of shared key cryptography to establish secure communication channels [3]. The key exchange problem describes ways to exchange keys and other information that are needed to establish secure communication channels. We can say that a KEX-protocol is called secure if it is infeasible for an adversary to distinguish the value of a key generated by the protocol from a random value.

In the following sections we analyse the security of the Diffie-Hellman key exchange protocol. The Man-in-the-Middle attack is the common attack that will be taken into consideration during the following sections, with a passive and active adversary. The basic idea behind Man-in-the-middle attack is that an adversary have full control of the communication channel and can violate a given security property, for instance confidentiality or authenticity [1], i.e an active adversary can attempt to impersonate both participants.

2.1 Diffie-Hellman

The Diffie-Hellman key exchange algorithm deals with the following dilemma. Alice and Bob want to share a secret key for use in a symmetric cipher, however their communication line is insecure. Information exchanged between Alice and Bob is observed by an

adversary Eve. How can Alice and Bob share a key without making it available to their adversary? Diffie and Hellman suggested the notion of *public key encryption*. Firstly the protocol is presented, subsequently the security goals are explained.

We use the convenience of a *cyclic group* $\mathbb{Z}_p^*$ for a prime p and a generator g. For every element $x \in \mathbb{Z}_p^*$, there is a $i \in \{0, ..., p-2\}$ such that x = g^i modulo p. i is called the *discrete log of x* with respect to g

2.2 The Diffie-Hellman key exchange protocol

- Alice and Bob agree on a large prime p and a generator g
- Alice chooses $a \xleftarrow{\$} \mathbb{Z}_p^*$ and computes $X = g^a$ modulo p. Alice sends X to Bob.
- Bob chooses $b \xleftarrow{\$} \mathbb{Z}_p^*$ and computes $Y = g^b$ modulo p. Bob sends Y to Alice.
- Alice computes $K_A = Y^a$ modulo p, and Bob computes $K_B = X^b$ modulo p
- Finally they use K_A and K_B as key to exchange messages using a private key encryption scheme.

A protocol diagram is given in Figure 2.1. The Diffie-Hellman protocol is susceptible to two attacks: *the discrete logarithm attack* and *the man-in-the-middle attack*. When defining security, it is important to keep in mind the anticipated adversary. For this section, we will focus on passive adversaries. A passive adversary can eavesdrop on the communication channel and learn information such as g, X and Y without interfering with the protocol.

The discrete logarithm attack: The adversary Eve, can intercept the values X and Y, and from these values find a from $X = g^a$ and b from, $Y = g^b$. Then she can calculate $K = g^{ab}$ modulo p. If the adversary can obtain K then the key is no longer secret. Let's make a proof that is more rigorous, by introducing a few assumptions:

Definition 2.1 - Discrete Logarithm (DL) assumption:
For a group $\mathcal{G}$ with $\langle g \rangle$, it is hard to compute a given a random eleven g^a.
Definition 2.2 - (Computional) Diffie-Hellman (DH) assumption:
For a group $\mathcal{G}$ with $\langle g \rangle$, it is hard to compute g^{ab} given random elements g^a,g^b.

Definition 2.3 - (Decisional) Diffie-Hellman (DDH) assumption:
For a group $\mathcal{G}$ with $\langle g \rangle$, it is hard to distinguish g^{ab} from a random group element g^c, given random group elements g^a,g^b.

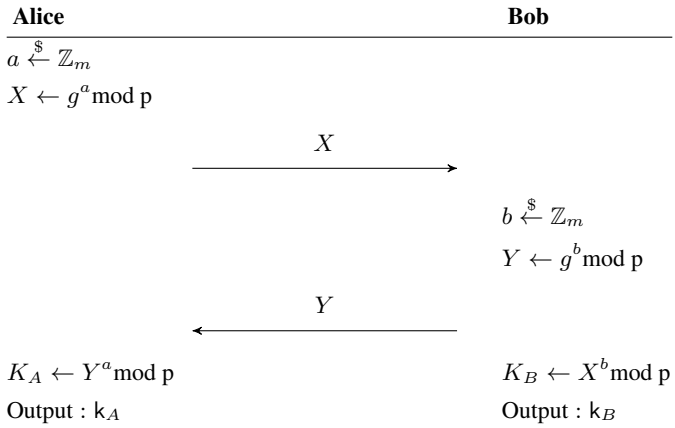


Figure 2.1: The Diffie-Hellman key exchange.

If you have a machine which can efficiently solve the DL-problem in some group, you can easily construct a solver for the DH-problem. If you have a solver for the DH-problem, you can also decide the DDH-problem. This means that, DDH is at least as easy, or easier, as DH, and DH is at least as easy as DL. Or for the other way around, DL is at least as hard as DH, and DH is at least as hard as DDH. Another way of looking at it is the assumption to be hard, where *hard* can be defined as not doable for a reasonable user. If DDH is hard, then DH must be hard too. If DH is hard, then DL must be hard. Thus, hardness of DDH is a stronger assumption than the hardness of DH, which is still stronger than DL.
So, evidently we see that DDH $\Rightarrow$ DH $\Rightarrow$ DL.

As previously mentioned, a passive adversary can obtain $X = g^a$ and $Y = g^b$ by eavesdropping on the communication channel. Let's say the adversary try to compute $a = \log_g X$. Success would contradict the DL assumption (Definition 2.1). Similarly for computation of $b = \log_g Y$. If an adversary $\mathcal{A}$ can find K from a and b, then we can use $\mathcal{A}$ to solve the DH-assumption. Under the Diffie-Hellman assumption (Definition 2.2), adversary $\mathcal{A}$ cannot find key K given a and b. However, if K is used to encrypt small messages, say a 1-bit message $m \in \{0, 1\}$, where ciphertext $C = g^M K$, m can be recovered deterministically from the quadratic residuosity of a and b. To avoid this problem, we introduce the Decisional Diffie-Hellman assumption (Definition 2.3) to exclude leak of

partial information. We can look into the security under DDH assumption.

Let f be a balanced function, that is

$$f : \langle g \rangle \ \Rightarrow \ \{0, 1\}$$

$$\Pr[f(u) = 0] = \Pr[f(u) = 1] = \tfrac{1}{2} \quad \text{for} \quad u \ \in \ \langle g \rangle$$

Suppose that $\mathcal{A}$ guesses $f(K)$:

$$\Pr[\mathcal{A}(a, b) = f(K)] > \ \tfrac{1}{2} + \varepsilon$$

for a non-negligible ε.
Let's define a distinguisher *D* for DDH:

$$D(\varphi, g^b, g^c) := \begin{cases} 1, & \mathcal{A}(g^a, g^b) = f(g^c) \\ 0, & \mathcal{A}(g^a, g^b) \neq f(g^c) \end{cases}$$

DH-triples:
$\Pr[D(a, b, K) = 1]$
= $\Pr[\mathcal{A}(a, b) = f(K)]$
> $\frac{1}{2} + \varepsilon$

Random triples:
$\Pr[D(a, b, g^c) = 1]$
= $\Pr[\mathcal{A}(a, b) = f(g^c)]$
> $\frac{1}{2}$

We see that the advantage of D exceeds the advantage of $\mathcal{A}$, and this contradicts with the Decisional Diffie-Hellman assumption.

Let DDH be the set of tuples $(g^a, g^b, g^{ab})|$ where $a, b \in 0, 1, ..., p - 1$.

Definition 2.4:
Let A be a CDH-algorithm for a group $\mathcal{G}$ in probabilistic time algorithm, for a fixed $\alpha > 0$ and a large n, satisfy:

$$\Pr[A(p, g, g^a, g^b) = g^{ab}] > \frac{1}{n^\alpha}$$

where g is the generator of the group $\mathcal{G}$.

Definition 2.5:
Let A be a DDH-algoritghm for a group $\mathcal{G}$ in probabilistic time algorithm, for a fixed $\alpha >$ 0 and a large n, satisfy:

$$\Pr[A(g, p, g^a, g^b, g^{ab}) = \text{"True"} - \Pr[A(g, p, g^a, g^b, g^c) = \text{'True'}] > \frac{1}{n^\alpha}$$

where g is the generator of the group $\mathcal{G}$. The difference between the two probabilities above is often called the *advantage* of algorithm A [2]. The definition captures the notion that the distributions p, g, g^a, g^b, g^{ab} and p, g, g^a, g^b, g^c are *computationally indistinguishable*.

One approach to make this protocol safe from this attack is by choosing some of the parameters with more care. The following are recommended [1]:

1. The prime number *p* must be very large, preferably more than 300 digits
2. The generator *g* must be chosen from the group $\langle\ \mathbb{Z}_p^*, x\ \rangle$
3. a and b must both be large and random numbers of at least 100 digits long, and used only once

Still, no algorithm for the discrete algorithm problem exists with computational complexity $O(x^r)$ for any *r*, all are unfeasible [9].

Chapter 3

Authenticated Key Exchange

Now that we have a secure protocol for passive adversaries, what happens with our security when we have an active adversary? The Diffie-Hellman protocol doesn't provide authentication of the communicating parties, meaning that a man-in-the-middle attack is possible. Thus, it is essential to use authentication and key establishment together. A protocol that achieves this is called authenticated key establishment (AKE). KEA is a Diffie-Hellman based key-exchange protocol that provides mutual authentication for the parties, which will be discussed in Chapter 4. AKE allows parties to compute the shared key while also ensuring authenticity of the parties. This means that a party can compute a shared key only if it is the one it claims to be. A natural solution for a authenticated key exchange protocol is to execute a *Signed Diffie-Hellman* key exchange, and sign all the communication sent between the parties.

3.1 Signed Diffie-Hellman

A digital signature is a cryptographic system used for verifying the authenticity of digital messages. Simply put, it is a way to validate the authenticity and integrity of any data. Digital signatures often use a public key encryption system. Consider the following: *How can Bob be sure that it was Alice who sent the message, and not an adversary pretending to be Alice?* As we have seen, the Diffie-Hellman protocol is sensitive to a man-in-the-middle attack, where the adversary runs the protocol separately with Alice and Bob. Alice and Bob are not able to distinguish each other's bits from Eve's bits, which can compromise

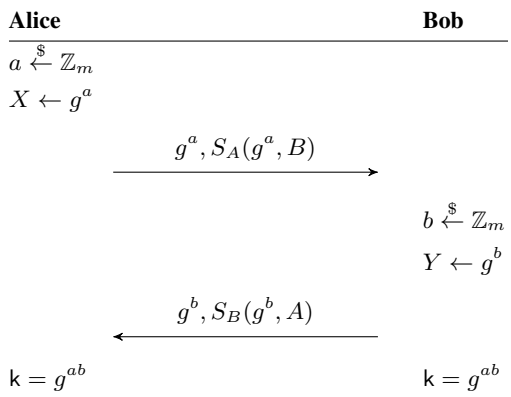


Figure 3.1: A first attempt signed Diffie-Hellman authenticated key-exchange.

the protocol. With the use of a digital signature it can provide authentication in their conversation. Before encrypting the message to Bob, Alice can *sign* the message using her private key. When Bob decrypts the message, he can verify Alice's signature by using her public key. Let's illustrate such a signature key exchange protocol by using a two passes protocol, that was suggested by Shoup [8]. Let $\mathcal{G}$ be a group of order p and let g be the generator. Denote the signature of a message M under the secret key of a party **A** as $S_A(M)$. This protocol has 2 passes, where an initiator **A** picks a secret key a at random and sends to a responder **B** a tuple $g^a, S_A(g^a, \mathbf{B})$. The responder **B** picks a secret key b and replies with the tuple $g^b, S_B(g^b, \mathbf{A})$. The parties will then verify each other's signatures and if accepted, compute a shared session key $K = g^{ab}$. The protocol is illustrated in Figure 3.1.

3.2 Matching Conversations

If we remove the encryption on the signatures, then the protocol will become insecure [8]. If we consider an adversary $\mathcal{A}$ controlling a different identity. Then $\mathcal{A}$ could generate its own signature, if the protocol did not have the encryption on the last message. This would result in a situation where **B** "thinks" he is talking to $\mathcal{A}$, but in fact shares a key with **A**, who "thinks" he is talking to **B**.

A security requirement that was introduced by Bellare and Rogaway [1] considers a multi-

Send(U,s,M)	Send message M to oracle $\prod_U^s$
Reveal(U,s)	Reveal session key (if any) accepted by $\prod_U^s$
Corrupt(U,K)	Reveal state of U and set long-term key of U to K
Test(U,s)	Attempt to distinguish session key accepted by oracle $\prod_U^s$

Figure 3.2: Representation of queries used in the Bellare-Rogaway model.

party experiment with unauthenticated communication channels. Bellare and Rogaway published the first mathematical proof that a simple entity authentication protocol was secure. They define a protocol where an adversary controls all communication between the participating parties and also interacts with the algorithms in the protocol offered by the cryptographic game. This section will give an informal definition of the Bellare-Rogaway model (BR-93, in short). The adversary selects honest parties to take part in key-exchange sessions. The adversary must select a test session and then be given a challenge, which is either the session key of the test session or a randomly selected key. The adversary's goal is to distinguish between these two cases. Within the field of security, an adversary refers to an attacker that undertakes an attack on a system or protocol. The adversary, very often, has malicious intents where the goal is to disrupt or prevent proper operation of a secure system. The BR-93 model allows different types of adversaries, e.g passive and active attacker. It also includes different queries, as shown in Figure 3.2, that we can see as the adversary capabilities in the protocol. Bellare and Rogaway define mutual authentication (MA) through an experiment with the running of adversary $\mathcal{A}$. After the session is terminated by $\mathcal{A}$, each oracle $\prod_{i,j}^s$ has had a certain conversation $k_{i,j}^s$ with the adversary and reached a certain decision $\delta \in \{A, K, *\}$. The security of the MA protocol can be defined through the distribution on mentioned conversations and decision, in what Bellare and Rogaway call *good* or *bad* executions. For any oracle $\prod_{i,j}^s$, let the following sequence capture its conversation:

$$\Lambda = (\tau_1, \alpha_1, \beta_1), (\tau_2, \alpha_2, \beta_2), \dots, (\tau_m, \alpha_m, \beta_m)$$

where the sequence encode that at time τ_1 oracle $\prod_{i,j}^s$ was asked α_1 and responded β_1; at a later time when $\tau_2 > \tau_1$ oracle was asked α_2 and responded β_2 and so forth until time τ_m where the adversary $\mathcal{A}$ stop asking any further questions and terminates the sessions. The introduction of more notions are required to further explain mutual authentication, for this we need to explain the notion of *an initator* and *a responder* oracles. Let oracle $\prod_{i,j}^s$ have the sequence $(\tau_1, \alpha_1, \beta_1)$; if $\alpha_1 = \lambda$ we call oracle $\prod_{i,j}^s$ the *initiator oracle*. If α_1 is any other string we call $\prod_{i,j}^s$ the *responder* oracle. Let the R be odd, $R = 2q - 1$, where

R is the number of moves. Let $\prod$ be the R-move protocol in the presence of an adversary $\mathcal{A}$ and two oracles $\prod_{A,B}^{s}$ and $\prod_{B,A}^{t}$ that accedes in conversations K and K', respectively.
Definition 3.1: We say that K' is a *matching conversation* with K if there exist $\tau_0 < \tau_1 < \ldots < \tau_R$ and $\alpha_1, \beta_1, \ldots, _q, \beta_q$ such that K is prefixed by

$$(\tau_0, \lambda, \alpha_1), (\tau_2, \beta_1, \alpha_2), (\tau_4, \beta_2, \alpha_3), \ldots, (\tau_{2q-4}, \beta_{q-2}, \alpha_{q-1}), (\tau_{2q-2}, \beta_{q-1}, \alpha_q)$$

and K' is prefixed by

$$(\tau_{1,1}, \beta_1), (\tau_3, \alpha_2, \beta_2), (\tau_5, \alpha_3, \beta_3), \ldots, (\tau_{2q-3}, \alpha_{q-1}, \beta_{q-1})$$

We say that K is *matching conversation* to K' if there exist $\tau_0 < \tau_1 < \ldots < \tau_R$ and $\alpha_1, \beta_1, \ldots, _q, \beta_q$ such that K' is prefixed by

$$(\tau_{1,1}, \beta_1), (\tau_3, \alpha_2, \beta_2), (\tau_5, \alpha_3, \beta_3), \ldots, (\tau_{2q-3}, \alpha_{q-1}, \beta_{q-1}), (\tau_{2q-1}, \alpha_q, *).$$

and K is prefixed by

$$(\tau_0, \lambda, \alpha_1), (\tau_2, \beta_1, \alpha_2), (\tau_4, \beta_2, \alpha_3), \ldots, (\tau_{2q-4}, \beta_{q-2}, \alpha_{q-2}), (\tau_{2q-2}, \beta_{q-1}, \alpha_q)$$

The first two sequences explains if the conversation of a responder oracle matches the conversations of an initiator oracle. The last two sequences defines when the conversation of an initiator oracle matches the conversation of a responder oracle. In other words, if every message sent out by $\prod_{A,B}^{t}$, the initiator oracle, is subsequently delivered to $\prod_{B,A}^{t}$, and the response is returned to $\prod_{A,B}^{t}$, then we say that the conversation of $\prod_{B,A}^{t}$ matches that of $\prod_{A,B}^{t}$, and similarly the other way around.

A notion that will prove to be very helpful in the following section is the notion of *partner*. The *partner* of an oracle proves itself to be a crucial element when defining security. Partners have been defined as having the same session identifiers which consists of a series of messages exchanged between the two. Partners must both have accepted the same session key and recognise each other as partners. However, the definition given from Bellare and Rogaway [1], in the presence of an adversary as powerful as the one they define it is unclear what it could mean to be convinced that one has engaged in a conversation with a specified partner. This because, every bit communicated has been communicated to the adversary instead. The **Test** query may only be used for an oracle which has not been corrupted and that has accepted a session key that has not been revealed during a **Reveal** query.

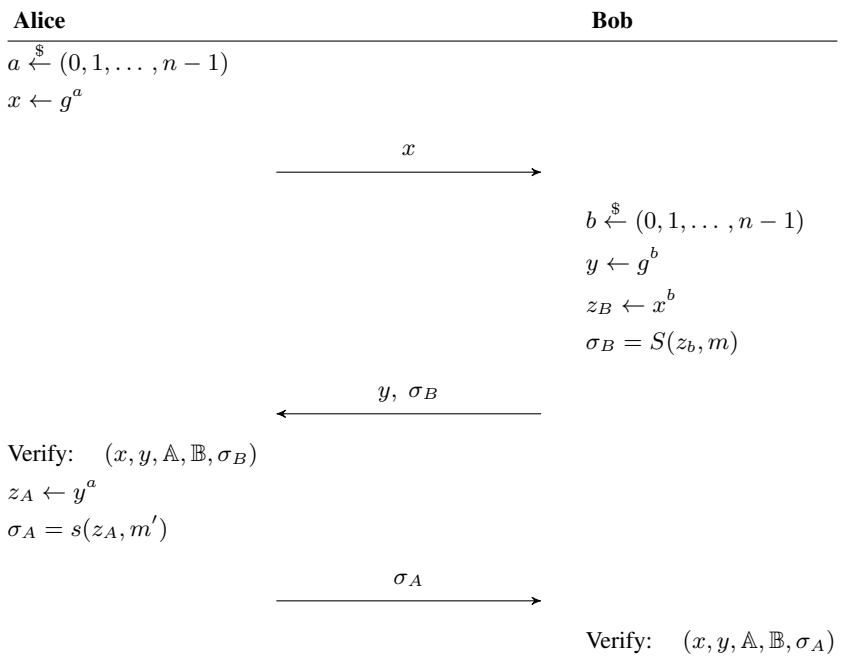


Figure 3.3: Digital Signature Protocol with standard Diffie-Hellman communication.

In addition, the partner of the oracle to be tested must not have had a **Reveal** query. To further look at the security requirements, we're going to present three games using the Bellare-Rogaway model. Each game will look and act very similar, although there are some small changes between them. A detailed description of each game will then be followed by an explanation of their difference. As mentioned, the games run using the BR-93 model where the adversary interacts with the oracles through queries. We assume it has full control over the communication network through a **Send** query which allows it to send arbitrary messages to any oracle. There are additional queries that the adversary is allowed to use shown in table 3.2. A simulator will receive the queries from the adversary and we let the simulator run the games between the oracles. The oracles will run a standard Diffie-Hellman communication.

Introduction Before introducing the games, we will give a formal description of how the games run. The oracles will either take the role as an *initiator* or a *responder*, depending on the query sent by the adversary. We let $R_1, R_2, \ldots, R_n$ be the number of all

responder oracles, and for each responder oracle we have a player P_j. Player P_j has a conversation with another player P_i. During the conversations we denote the initiator with the following notation $I/i/P_i/P_j$, where player P_i is the initiator I and has a conversation with player P_j. Conversely, the responder oracle is denoted $R/j/P_j/P_i$ where player P_j is the responder R and has a conversation with player P_i.

Game 0 The standard game where the oracles use standard Diffie-Hellman communication with signatures in the Bellare-Rogaway model. The initiator $I/i/P_i/P_j$ select ephemeral public keys x_i at random and sends it to the responder $R/j/P_j/P_i$. The responder also select the public key y_i and sends it back to the initiator. Initiator must verify the signature and message from the responder and they agree on a secret key k_i, as shown in Figure 3.4.

Game 0 I/i/P$_i$/P_j	Game 0 R/j/P$_j$/P_i
1 : $a_i \xleftarrow{\$} (0, 1, \ldots, n-1)$	1 : $b_i \xleftarrow{\$} (0, 1, \ldots, n-1)$
2 : $x_i \leftarrow g^{a_i}$	2 : $y_i \leftarrow g^{b_i}$
3 : **Send** x_i	3 : **Send** y_i
4 : **Get** $(y_i, \sigma_{R,j})$	4 : **Get** $(y_i, \sigma_{R,j})$
5 : $\sigma_{I,i} \leftarrow$	5 : $\sigma_{R,j} \leftarrow$
6 : **Send** $\sigma_{I,i}$	6 : **Send** $\sigma_{R,j}$
7 : $k_i \leftarrow \mathsf{H}(y_i^{a_i})$	7 : $k_j \leftarrow \mathsf{H}(x_i^{b_i})$

Figure 3.4: A representation of *Game 0*.

Game 1 This game runs the same steps as *Game 0*, however with a small change. In *Game 0* we draw $a_i, \ldots, a_N, b_i, \ldots, b_N$ at random, while in this game the simulator makes sure that they are distinct. The initiator check that $a_i \neq a_k \forall i, k$ and the responder checks that $b_i \neq b_k \forall i, k$. For every Alice that is finished, there exists a Bob that is talking to Alice and they have a matching conversation. To summarize the above we only change **Step 1** from *Game 0*, however the other steps remain the same.

Game 1 R/j/P$_j$/P_i
1 : $a_i \xleftarrow{\$} (0, 1, \ldots, n-1) \backslash (a_{1,2}, \ldots, a_i)$

Modification We observe that this game has a small change from our previous game, but why isn't the game above different from *Game 0* ? We previously drew $a_i, \ldots, a_N, b_i, \ldots, b_N$

at random, however in this game they are still random but *distinct*. In essence, we only make sure that $a_i \neq a_k$ and $b_i \neq b_k$ in *Game 1*. From the *birthday paradox* we know that if we draw n different values and if we draw n random values, we really can't tell the difference if we don't draw to many values. From this we can't tell the difference between the two games.

Consequences The consequence after changing the first step in *Game 0* is that it disallows that for some $\prod_i^s$ there exists distinct oracles $\prod_i^t$ and $\prod_i^{t'}$, such that $\prod_i^s$ is a partner to both $\prod_i^t$ and $\prod_i^{t'}$, this would break *uniqueness*.

Comments The modification done in this game, $a_i \neq a_k$ and $b_i \neq b_k$, would in reality never be possible because *Person A* would not be able to know what random number *Person B* chose. However since the game runs with the BR-93 model, using a simulator, we are able to accomplish this. From the *birthday paradox* we know that we don't see the difference between the two, so we can't tell the difference between the games.

Game 2 In this game we want to check that there is no forgery in the signatures. This game does also run as the ones above, however we skip the verification step. We omit the signature and verification, and only check if it truly was Bob that sent y and $\sigma_{R,j}$. This means that we only change *Step 4*. The players run the Signed Diffie-Hellman as before and whenever player $I/i/P_i/P_j$ obtain a signature $\sigma_{R,j}$ from player $R/j/P_j/P_i$, it will ignore the signature and message if it was not sent from that player.

Modification We argue that this change is still not observable, and that it doesn't make the game different from the previous one. However this is a more drastic difference in steps so far, how does is still not make the game different? Firstly, our only objective in this step is to verify if the signature was truly sent by a responder oracle, and is not a forged signature. From the protocol we know that if Alice accepts the query she will send x to Bob, and receive y and $\sigma_{R,j}$ back. She will continue to reply with $\sigma_{I,i}$ and agree on a key. If we don't have a forged signature, then Alice will produce a key, which means that the responder oracle was the one sending y and $\sigma_{R,j}$,and there exists a matching conversation between the players. We intend to make a reduction to prove that this is an observable change for the adversary.

Reduction In our reduction, we hope that the valid signature is made precisely for player k because then we would have obtained our signature. We let the simulator pick verification key vk_k from player k, and let $vk_k = vk$. Whenever the adversary needs to make a signature, the adversary sends a message m_i and receives a signature σ_i from the simulator. The adversary's objective is to produce (m', σ') that is a valid signature σ' for message m'. The adversary has the opportunity to send query many times and for each time he runs player k, as an initiator or responder, he must create a signature. Conclusively, we receive our (m', σ') that was not created by the simulator. We can easily verify the signature because we already have access to the verification key vk. This means that the only way that the change in *Step 4* is noticeable is if we get a (y, σ) that is valid and not made by the simulator. However, this would imply that we have an adversary that is capable to produce forged signatures. Our reduction is presented in Figure 3.5.

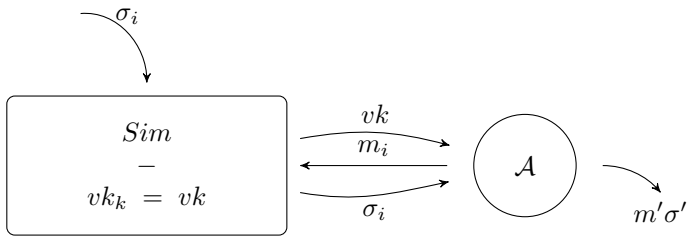


Figure 3.5: Reduction from *Game 0*.

Consequences What we are able to establish with the (unobservable) change done in this game, is the notion of partnering, and we obtain a *passive DH* that we've previously proven secure. From figure 3.7 we have an adversary against key exchange denoted A_{KEX} and a simulator denoted Sim_{KEX}. In the reduction from *Game 2* we also have an adversary against signatures, which we will be denote $\mathcal{A}$ and a simulator Sim shown in Figure 3.5. Let F_i be the case where we receive a signature that was not produced from an oracle in game i. Let E be the case where $b = b'$ in game i, and let H_i be the case where the group elements that the oracles pick in *Step 1* are unique in game i. We begin by looking at the advantage for the adversary against key exchange, where

$$\begin{aligned}
\text{Adv}A_{\text{KEX}} &= \left|\Pr[\text{E}_0] - \frac{1}{2}\right| \\
&= \left|\Pr[\text{E}_0] - \Pr[\text{E}_1]\right| \\
&+ \left|\Pr[\text{E}_1] - \Pr[\text{E}_2]\right| \\
&+ \left|\Pr[\text{E}_2] - \frac{1}{2}\right|
\end{aligned}$$

Next we want to further look at the statement done in *Game 1* where modification was that the simulator picks element $a_i \neq a_k$ and $b_i \neq b_k$. We look at the following equations

$$E_0|H_0 = E_1|H_1$$
$$\big|\Pr[E_0] - \Pr[E_1]\big| \leq \Pr[\neg H_0] \leq \frac{n^2}{2|G|}$$

which means that

$$H_0 : \ \Pr[H_0] \geq 1 - \frac{n^2}{2|G|}$$
$$H_1 : \ \Pr[H_1] = 1$$

This states the fact that the group elements chosen in *Game 1* are all unique and different, which we have already argued for using the *birthday paradox*. Now, what we are really interested in proving in *Game 2* is that the probability is low to observe a forged signature and therefore the game is not noticeably different from the previous ones. We argue that the probability for this is

$$\begin{aligned}\mathrm{Adv}_{\mathrm{Sig}}B_{\mathrm{Sig}} &= \frac{1}{n}\Pr[\mathrm{G}_i] \\ &= \frac{1}{n}\Pr[\mathrm{F}_1] \\ &= \frac{1}{n}\Pr[\mathrm{F}_2]\end{aligned}$$

Where G denotes all games i, and we wish to prove that the probability is the same for *Game 1* and *Game 2*. Firstly, we know that

$$\Pr[\mathrm{E}_2|\neg\mathrm{F}_2] = \Pr[\mathrm{E}_1|\neg\mathrm{F}_1]$$

in *Game 1* and *Game 2*. By this, we can finally begin to conclude the following probability

$$\begin{aligned}\big|\Pr[\mathrm{E}_2] - \Pr[\mathrm{E}_2]\big| &= \big|\Pr[\mathrm{E}_2|\mathrm{F}_2] \cdot \Pr[\mathrm{F}_2] + \Pr[\mathrm{E}_2|\neg\mathrm{F}_2] \cdot \Pr[\neg\mathrm{F}_2] \\ &\quad - \Pr[\mathrm{E}_1|\mathrm{F}_1] \cdot \Pr[\mathrm{F}_1] - \Pr[\mathrm{E}_1|\neg\mathrm{F}_1] \cdot \Pr[\neg\mathrm{F}_1] \\ &= \Pr[\mathrm{F}_1]\big|\Pr[\mathrm{E}_2|\mathrm{F}_2] - \Pr[\mathrm{E}_1|\mathrm{F}_1]\big| \\ &\leq \Pr[\mathrm{F}_1] \\ &= n \cdot \mathrm{Adv}B_{Sig}\end{aligned}$$

This means that this change is only noticeable if the adversary manage to forge a signature. And if that probability is small then we are we are not able to see the difference between games.

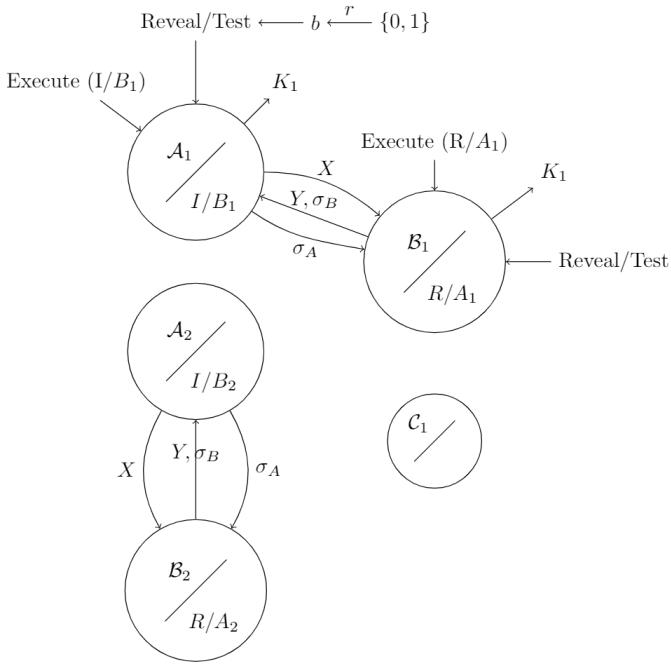


Figure 3.7: Algorithm on key exchange between oracles, in the presence of an adversary using queries on A_1 and B_1.

Conclusion We have presented three games that differ in a few steps, however proven that they still are not different from one another. We will finally look at a statement that we wish to prove as a conclusion to this chapter, as well as shining a light on two security notions that has been proved. We begin by proving the notion of *secrecy*, the adversary can't guess for a **Test** query. Remember that E_i is the event where $b = b'$.

$$\begin{aligned}
\left|\Pr[\mathrm{E}_0] - \Pr[\mathrm{E}_1]\right| &\leq \varepsilon \\
\left|\Pr[\mathrm{E}_2] - \Pr[\mathrm{E}_1]\right| &\leq \Pr[\mathrm{F}_2] \\
\left|\Pr[\mathrm{E}_0] - \frac{1}{2}\right| &= \mathrm{Adv}_A^{\text{Passiv-KEX}} \\
&\approx cAdv_B^{\text{DDH}}
\end{aligned}$$

where ε is some small negligible value.

Furthermore we want to look at the notion of matching conversations that was presented earlier and prove *authentication*. We can accomplish this by analysing the following event U that says

U : An oracle is finished and there are *no other* oracles with matching conversations
or there are *more than one* oracle with matching conversations

Let U_i be the event U in game i. When *Game 2* is finished we know that the event F has not occurred and there was not made a forged signature, which ultimately means that they all have a matching conversation. The probability for U_2 will then be

$$\Pr[\mathrm{U}_2] = 0 \tag{3.1}$$

and further,

$$\begin{aligned}
\Pr[\mathrm{U}_0] &= \Pr[\mathrm{U}_0] - \Pr[\mathrm{U}_1] + \Pr[\mathrm{U}_1] - \Pr[\mathrm{U}_2] + \Pr[\mathrm{U}_2] \\
&\leq \big|\Pr[\mathrm{U}_0] - \Pr[\mathrm{U}_1]\big| + \big|\Pr[\mathrm{U}_1] - \Pr[\mathrm{U}_2]\big| \\
&\leq \epsilon + \Pr[\mathrm{F}_2] + \Pr[\mathrm{U}_2] \\
&= \epsilon + \Pr[\mathrm{F}_2]
\end{aligned}$$

where ε is some small negligible value.

In addition to the probability of F_e being small we also know from (3.1) that the probability for U_2 is small. This reasoning, in addition to the fact that we have different and unique values in our public key X_i and y_i, we have established unique partnering. Which also fulfils the authentication requirement, and we have matching conversations and authenticated key exchange.

Chapter 4

The KEA protocol

4.1 Overview

In our previous section our goal was to investigate the security analysis in a Signed Diffie-Hellman key exchange. We wish to consider a new 2-pass key exchange protocol that continue to give authentication, however without the use of signatures. In this section our goal is to study the security of the KEA+ protocol, using the same approach with the BR-93. We wish to prove that both *authentication* and *secrecy of keys* are preserved in this protocol. We intend to divide our security analysis in two parts, where in *Part 1* we have honest players that either have matching conversations or have distinct keys and in *Part 2* we have an adversary that doesn't have the keys.

4.2 The KEA+ protocol

KEA was designed by NSA in 1994, although it did not become available to the public until 1998. KEA involves two parties, **A** and **B** with their respective secret keys a and b and public keys g^a and g^b. KEA protocol first execute a Diffie-Hellman communication where the parties select secret keys x and y and exchange the public keys g^x and g^y. Each party will compute g^{ay} and g^{bx}, then compute a session key *k* by applying hash function $H\big(g^{ay}, g^{bx}\big)$. However there are attacks available against the KEA protocol, the AKE security of KEA can be violated if an adversary can register arbitrary public keys [6]. An

example for an attack against the KEA protocol is *Unknown Key Share Attack*. A solution is to modify the protocol and make it resistant to such attack, and this is done in the KEA+ protocol. This protocol incorporate the parties identities in the computation of a session key. The parties compute the key k as follows $H\big(Id_A, Id_B, X, Y, g^{ay}, g^{bx}\big)$. This protocol is depicted in Figure 4.1.

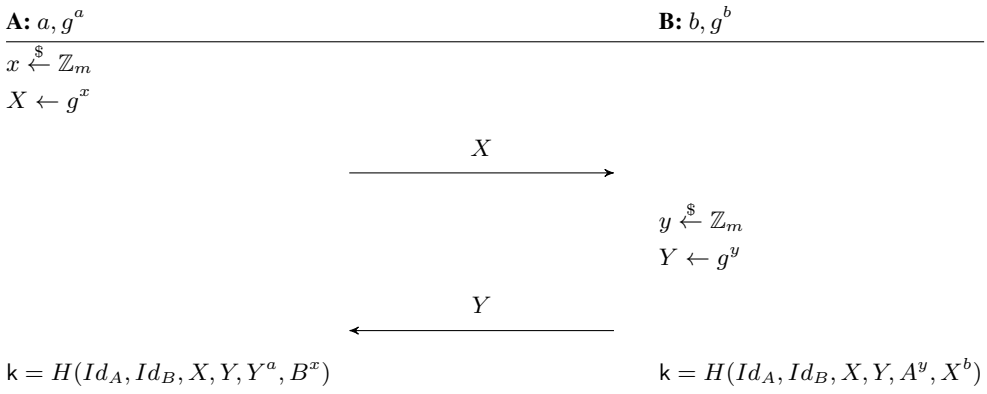


Figure 4.1: The KEA+ protocol.

This 2-pass protocol has exactly the same communication as the original Diffie-Hellman protocol, however this protocol doesn't provide delivery guarantees that would be desirable. In other words, it doesn't provide assurance that the other party actually completed the session, however we will address this case in this section.

As indicated formerly, we divide our security analysis in two parts. We introduce our first part, where we intend to prove that we have honest players that either have matching conversations or have distinct keys. So far we have looked at two variants of signed Diffie-Hellman which we have improved, and looked at the security notions. From Figure 4.1 we see that KEA+ is a 2-pass protocol, where there is very likely that we obtain many matching conversations. Whenever a player outputs X, many players can receive X and therefore have matching conversations. As stated above, there isn't provided some assurance that the other party has completed a session and have matching conversations.

However our main goal at this moment is to prove authentication. Let V be the event that

V : i) An oracle is finished with key k, and

ii) there are two or more oracles of the same role that is finished with key k, or

iii) there is one oracle that i finished with key k but doesn't have matching conversation

We will analyze event V by introducing two new games, denoted *Game 1* and *Game 2*. The games run the same BR-93 model, where the oracles communicate with one another, and take part as an *inititator* or *responder*. Let
Game 1: All keys k, X and Y sent between the parties must be different.
Game 2: We have no collision in H. This means that we are not able to obtain the same key k if we put different values in H.

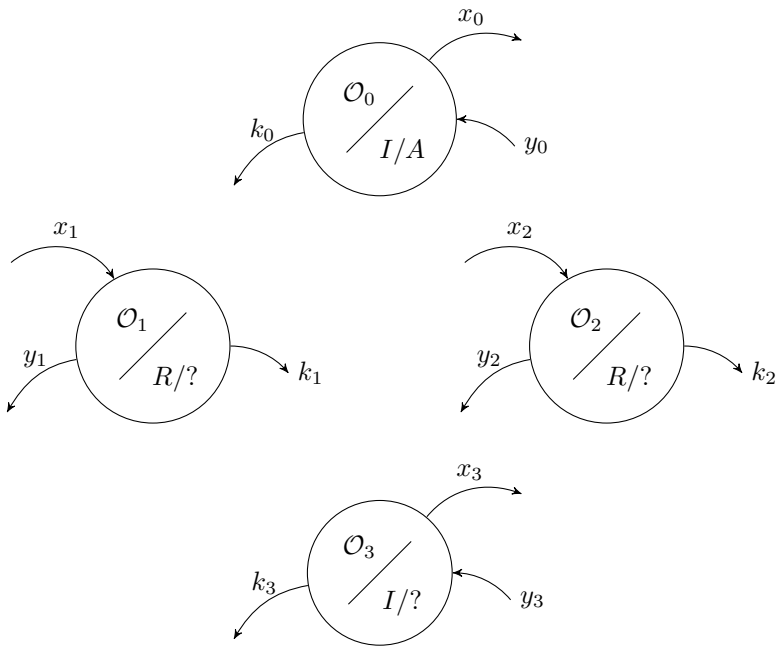


Figure 4.2: Representation of interaction between oracles in ii).

We initiate our analysis by considering $ii)$ in event V. The standard game contains an oracle O_0, as an initiator, that outputs x_0 and receives y_0 and produces a key k. To satisfy $ii)$ there must be two or more oracles of the same role that are finished with key k. We wish to further divide this point into two parts. One where we have the case of two reponder oracles O_1 and O_2, and a new case where we have another initiator oracle O_3. The key k

produced by the oracles is a hash function $H(Id_A, Id_B, X, Y, Y^a, B^x)$ from Figure 4.1. The oracles O_1 and O_2 acting as responders, have received a message x but are unsure of which player they're communicating with. These oracles also outputs their y's and produce a key k. In our last case, we have another initiator O_3 who outputs a message but is unsure of whom he is communicating with. Figure 4.2 depicts a simple representation on the communication between the oracles, where we observe that the oracles outputs distinct keys. However we ask the following two questions

$$\text{What if } k_0 = k_3?$$
$$\text{What if } k_0 = k_1 = k_2 = k_3?$$

If we allow the keys k_0 and k_3 to be identical, we automatically know that $x_0 = x_3$ because the keys are the same. The key k_0 is a hash of $H(Id_{O_1}, Id_{O_3}, X_0, Y_0, Y_0^a, B^x)$ and the key k_3 is a hash of $H(Id_{O_3}, Id_{O_0}, X_3, Y_3, Y_3^a, B^x)$ which evidently implies that if $k_0 = k_3$, it follows that $x_0 = x_3$ because *Game 2* doesn't accept collision which means that the input must be the same. In other words, we have established the following

$$k_0 = k_3 \implies x_0 = x_3 \wedge y_0 = y_3$$

By allowing the keys to be the same we also have settled a relation between the two initiator oracles and that is the case that $O_0 = O_3$, which means that we can't have two oracles as initiators. Further we look at the case where $k_0 = k_1 = k_2 = k_3$. So far we do not encounter any problems because the adversary is fully capable in making $x_0 = x_1$ or $x_0 = x_1 = x_2 = x_3$. However it is problematic with a collision with the y-values. Because if $x_0 = x_1 = x_2 = x_3$ it follows that $y_2 = y_1$ which implies that the oracle have the same y-values which we already have stated in *Game 1* that is not allowed and also established the relation that $O_1 = O_2$. To summarise we have stated the following,

$$x_0 = x_1 = x_2 = x_3 \implies O_0 = O_3$$
$$y_2 = y_1 \implies O_1 = O_2$$

This means that the $ii)$ can't happen, and we can't have two or more oracles of the same role that is finished with key k. We look further into analysing $iii)$ in event V, where we initially stated that an oracle that is finished with a key k doesn't have a matching conversation. Let O_1 be another oracle that acts as a responder, and receives x_1. It replies with y_1 and then produces a key k_1 that has the same hash function as the one from Figure 4.1. However this oracle doens't know which player it is communicating with, but let us

say for now that is not oracle O_0. Hence,

$$(O_0, O_2) \neq (O_1, O_3)$$

Where (O_0, O_2) denotes a conversations between O_0 and another oracle O_2. In addition, we also let

$$k_1 = k_0 \tag{4.1}$$

If O_0 doesn't have matching conversations, this would imply that

$$(O_0, O_2) \neq (O_1, O_3) \vee (x_1, y_1) \neq (x_0, y_0)$$

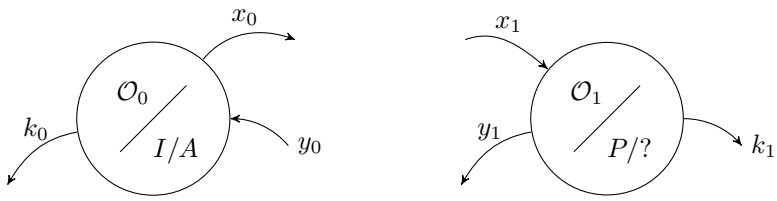


Figure 4.3: Representation of interaction between oracles in iii).

However for Figure 4.1 to be true, it would mean that the values in the hash function for k_1 must be same values in the hash function for k_0, since we have established in our games that two keys can't be the same with different values in their hash functions. This implies that the identities of the oracles that are inside the hash function must also be same as well as all other values. If this is the case, it means that O_0 *must* have a matching conversation with O_1 when it is finished with its key k. Statement $iii)$ in event V can not happen. To summarize the above, statement $ii)$ and $iii)$ in event V can't happen, which means that we have obtained *authentication* and concludes the proof in *Part 1* - we have honest players that either have matching conversations or distinct keys. We have so far not commented on or taken into account the *secrecy of keys*.

Our attention have mainly been on the hash function used in this protocol, and the notion of authentication comes trivially by all the values we have inside this function. Now we will focus on the secrecy of keys where we allow the adversary to use **Test** queries to get the key, and check whether he sees the difference between the two. By this commence to prove *Part 2* - that we have an adversary that doesn't have the keys. We introduce a new

event F, where we let

$$\text{F}: \text{ The adversary ask for } H(Id_A, Id_B, A, B, X, Y, B^x, Y^a)$$

Let A be a oracle that is an initiator and is communicating with B, that behaves the same way as Figure 4.3. However we stress that y *does not* come from a oracle with matching conversation. This means that y can either come from the adversary or the adversary has taken another oracles y and sent it to A. This is where authentication is important, because we want to prove that the adversary can't obtain k just by sending y to the oracle. We let the event E be the case where

$$\text{E}: \ b' = b$$

which is the same case from Figure 3.7. The probability for a simple guess from the adversary is given below and we are also interested in the probability for the adversary to guess E when he has asked for the hash function. In reality, we can assume that it would be close to 1.

$$\Pr[\text{E}|\neg\text{F}] = \frac{1}{2}$$

$$\begin{aligned}
\left|\Pr[\text{E}] - \frac{1}{2}\right| &= \left|\Pr[\text{E}|\text{F}] \cdot \Pr[\text{F}] + \Pr[\text{E}|\neg\text{F}] \cdot \big(1 - \Pr[\text{F}]\big) - \frac{1}{2}\right| \\
&= \left|\Pr[\text{F}]\big(\Pr[\text{E}|\text{F}] - \Pr[\text{E}|\neg\text{F}]\big)\right| \\
&= \left|\Pr[\text{F}]\big|\big(\Pr[\text{E}|\text{F}] - \frac{1}{2}\big)\right| \\
&\leq \frac{1}{2}\Pr[\text{F}]
\end{aligned}$$

4.3 Secrecy of keys

As stated above, we have previously not taken into account any possible tampering on y by the adversary, however focused on *implicit authentication* meaning that we are satisfied only on having matching keys k between the parties and obtain a matching conversation. Let k_0 be the key produced by oracle A with the hash function $H(id_A, id_B, A, B, X, Y, B^x, Y^a)$. Essentially we have known values X and B^x, however the value Y is sent by someone else, which could possibly be the adversary. The adversary wins if he can produce key k just by sending Y, and we wish to prove that this is not possible. In essence, we want to utilize the adversary to calculate a Diffie-Hellman problem, essentially the Decisional Diffie-Hellman problem. The problem we want to send to the adversary is the following tuple (X, B, W) and verify if he can deliver B^x, meaning $W = B^x$. Recall that the adver-

sary can ask many queries when he ask for H, because if event F happens then we know that one of the hash-queries is in the tuple. Nonetheless we do stumble on a problem; Decisional Diffie-Hellman states that the he can't tell the difference between g^c and g^{ab}.

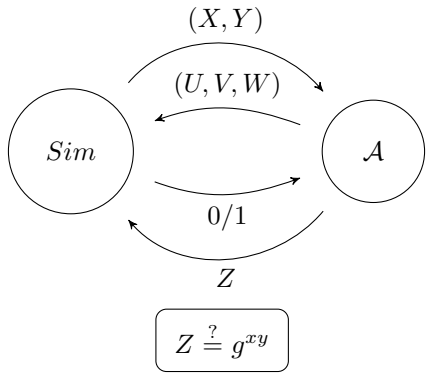


Figure 4.4: A depiction of the Gap oracle.

A solution *The Gap problem*

The solution to our problem is to utilize *gap problems* or *gap Diffie-Hellman*, and use this *Gap oracle* whenever our event F happens. In this game we are allowed to recognize the right answer. A gap problem is to solve a decision problem with the help of an oracle. An example for such a problem is given a problem x and a relation f where the adversary's goal is to find y satisfying $f(x, y) = 1$. With the use of an oracle for help, it can send (x', y') and get response of whether $f(x', y') = 1$ or not [7]. In our model we have a *Gap Oracle* that we have access to, that can help us identify real DDH-tuples. As previously used in our other game we have a simulator, denoted Sim_{KEX} that runs the conversations between players, and we wish to prove the secrecy of keys from the adversary $\mathcal{A}$. Both $\mathcal{A}$ and Sim_{KEX} have access to an oracle that outputs the hash value of an input r computed $r = (Id_A, Id_B, A, B, X, Y, W, R)$ and $W = Y^a$ and $R = X^b$. Sim_{KEX} calculates r by using both X and Y sent between the players.

The Gap Oracle In our game there is an interaction between a simulator and an adversary $\mathcal{B}$ where the objective is to produce Z where $Z = g^{XY}$, as shown in Figure 4.4. The adversary is allowed to use a **Test** query where it sends a tuple (U, V, W) and receives 0

or 1, where

$$1 \text{ if } \log_W = \log_U \log_V$$
$$0 \text{ else}$$

Whenever the adversary asks for the hash function for k, i.e. $H(id_A, id_B, A, B, X, Y, W, R)$ where $W = Y^a$ and $R = X^b$, we can simply take our tuple of (X, B, W) and run it in our Gap oracle. We obtain our Z if the adversary wins in Gap oracle, which also means that the adversary has managed to produce the hash function for k. However this also means that we break the Diffie-Hellman problem when we manage to notice $Z = g^{ab}$, so whenever the adversary wins we also win.

The model Our model will run a standard random-oracle model, which essentially means that our oracle will pick a $H(r)$ at random, for every r received from either $\mathcal{A}$ or Sim_{KEX}. The oracle will register and store every pair $(r, H(r))$. In addition to this, it will also check r and add $1/0$ if W and R are correct.

$$(r, h, 1) \text{ if } W = Y^a \text{ and } R = X^b$$
$$(r, h, 0) \text{ else}$$

To check whether r is correct, we can simply send this quest to the *Gap Oracle* and check if the (A, Y, W) and (B, X, R) are correct DDH-tuples. The oracle will check for every r if this has been asked before, if so it will output the corresponding $H(r)$. If not, it will pick a new random value and output this. So far this model is rather simple where the random oracle mainly store pairs and check whether new inputs has been previously asked.

We will make a small change to this model. Sim_{KEX} will omit W and R in further requests for $H(r)$, which means that it will send r with all its previous values apart from the last two that it will leave blank. The random oracle will function as previously and store all submissions if not received at a prior occasion. On any occasion where $\mathcal{A}$ sends r, the oracle will check with the Gap Oracle if it is correct and if it is stored. If it is correct and stored, it can substitute r with the values that were initially blank by the simulator with the correct W and R sent by $\mathcal{A}$. If the received r is not correct it will yet again pick $H(r)$ at random and output this value. The small change done in our model is the simple issue that Sim_{KEX} doesn't calculate any keys and the random oracle store all submissions independent if they're correct or not.

With this model we can say that if $\mathcal{A}$ wins, then we can win the *gap problem*. $\mathcal{A}$ can send

Test queries to the oracles where the objective is to obtain the right k. The description above is only the machinery that takes place in our game, Figure 4.5 depicts the game more detailed. We receive $\tilde{X}$ and $\tilde{Y}$ from the Gap Oracle. When received r from $\mathcal{A}$ we use the same oracle to verify for correct values, and lastly $\mathcal{A}$ outputs a b'. We're ultimately dependent on which oracle the adversary is going to send its query to. We have three different cases of where $\mathcal{A}$ sends **Test** query. *Firstly* it can send a query to a responder oracle that doesn't have matching conversations. *Secondly* it can ask a query to an initiator that doesn't have a matching conversation and *thirdly* it can ask a query to an oracle that does have a matching conversation. Ultimately we can only hope for and guess which oracel that gets asked the **Test** query by $\mathcal{A}$ and assign input that we have control over.

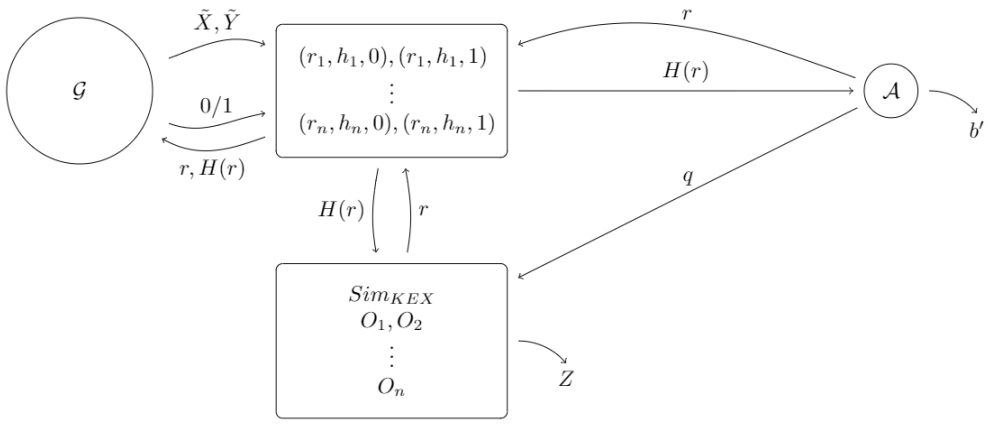


Figure 4.5: An illustration of the communication between $\mathcal{A}$ and Sim_{KEX} with the Hash Oracle and Gap Oracle.

We look at the case where the query is sent to a responder that doesn't have matching conversations. In this case we have control over Y which essentially means that we have the public key B, while the input X is sent by $\mathcal{A}$ which means that we don't have access to neither R nor X^b. In addition, we have control over public key A and also W since $W = Y^a$, which means we have the DDH-tuple (A, Y, W) so we let A be $\tilde{X}$ and Y be $\tilde{Y}$. We know the hash query is $k = (Id_A, Id_B, A, B, X, Y, W, X^b)$ even though we don't know how to calculate the value of W, because it is the answer to the DDH-problem. However we know that k is calculated using this hash query, and if $\mathcal{A}$ is able to distinguish it from a random value it means that the adversary has asked for the hash value and essentially given us W, and lastly let $\tilde{Z}$ be W.

Now we can approach the second case where the oracle is an initiator without a matching conversation. In this case we have control over X and therefore let X be our $\tilde{X}$ and let B be $\tilde{Y}$. The value k is the same hash as above, where it initially consists of two blank spaces for W and R since the simulator doesn't calculate these values. We use Gap Oracle whenever $\mathcal{A}$ outputs a key k and check if it contains the correct DDH-tuple. In this case it is R that we don't know since $W = Y^a$ and we don't have control over Y. We can use the same argument as our previous case. If $\mathcal{A}$ is able to distinguish k from a random value it means that the adversary has asked for the hash value and essentially given us R, and let $\tilde{Z}$ be R. The last case is easy to prove. In this case we let X be $\tilde{X}$ and the public key B be $\tilde{Y}$, since we have control over both X and Y.

Summary $\mathcal{A}$ can send **Test** query to an oracle and expect k back. The main question is which oracle receives the query, and hope that we choose the right oracle to put $\tilde{X}$ and $\tilde{Y}$ in and we are certain that there are three cases to choose from, where the goal is that the adversary cannot see the difference between these cases. In which case $\mathcal{A}$ will either obtain a k that is calculated or a randomly chosen value. If $\mathcal{A}$ calculated the right hash query, then we can simply use this query to solve the Gap Diffie-Hellman problem. In addition, if event F doesn't happen, it means that the adversary has to see the difference between a chosen value and a chosen value, which is not possible.

We additionally intend to introduce a new adversary query, *Long-term key reveal* where $\mathcal{A}$ can ask for the players long-term key. If we add this query to our current protocol, we will lose our secrecy of keys. Our next section will introduce an improved protocol that will be secure against this attack.

Chapter 5

Forward Security

Forward secrecy, also known as perfect forward secrecy, is a feature of key agreement protocols that assures that session keys will not be compromised even if the private key is compromised. It protects past sessions against future compromises of secret keys that may happen. To establish forward security the objective is to not use single key, e.g private key, to generate all the sessions keys. In this section our intention is to improve the KEA+ protocol and obtain forward secrecy.

5.1 Improving the KEA+ protocol

We start this section by introducing a new attack against the previous KEA+ protocol. The communication in the Diffie-Hellman protocol and KEA+ are both similar, however the keys are calculated differently. Recall that the keys k_{DH} and k_{KEA+} are calculated

$$k_{DH} = g^{xy}$$
$$k_{KEA+} = H(Id_A, Id_B, X, Y, g^{ay}, g^{bx})$$

for Diffie-Hellman and KEA+ respectively. Both parties have a long-term private key, e.g. a and b, and initially know the public key of all other participants. So far we've seen many possible and strong adversary queries as explained in Figure 3.2, and our intention is to propose two new queries, *Long Term key reveal* and *Ephemeral key reveal*. We begin by introducing the *Long Term key reveal* adversary query, where $\mathcal{A}$ has the capability of

learning the long-term private keys. In this section we wish to make a new and secure version of the previous protocol with this new query. Immediately we come across a complication with this query with the key k_{KEA+} considering that the adversary obtains the values a and b that are both necessary to obtain k_{KEA+}. Thus our current change must be in the way our keys are calculated. We let $Fresh'$ be the requirement that no Long Term key reveal can be asked before the partners are finished with the protocol. This is a requirement necessary because for $Alice$'s case, it shouldn't matter if she lost her key. She should still be convinced that she is communicating with Bob. Therefore, a player can't be compromised before its partner is finished with the protocol, if so $\mathcal{A}$ breaks $Fresh'$ and can't use a **Test** query.

As previously stated, it is necessary to update the calculation of key k. Our contribution is to combine the keys k_{DH} and k_{KEA+} such that our new key k is calculated as follows

$$k = H(Id_A, Id_B, X, Y, g^{ay}, g^{bx}, g^{xy})$$

With this small change, we have a new protocol that is secure even with the new query. We intend to prove that this protocol is secure using the three cases explained in the previous section. In these cases we still let $\tilde{X}$ and $\tilde{Y}$ be the same values that we chose earlier.

$$Case\ 1.\ \tilde{X} = A, \tilde{Y} = Y$$
$$Case\ 2.\ \tilde{X} = X, \tilde{Y} = B$$
$$Case\ 3.\ \tilde{X} = X, \tilde{Y} = B$$

In our previous protocol we had to utilize the Gap Oracle, however in this protocol we only need to utilize plain Diffie-Hellman. It doesn't affect the players if $\mathcal{A}$ learns the long term keys a and b in this protocol, because it's still not possible to calculate the value g^{xy}, and we assume these three cases as secure against this attack.

This model in addition includes a new notion of an *Ephemeral Key Reveal* adversary query. So far our protocol fails to be secure under this new query, since X and Y can be revealed. When the adversary obtain X and Y it essentially has all the values needed to compute the current suggestion on key k. NAXOS is a protocol that builds on earlier ideas from the KEA and KEA+ protocols [4]. Its purpose was to establish shared symmetric keys between parties and for it to be secure in a very strong sense, the protocol should be secure even if the adversary either learns the long-term key of a participant or learns the short-term data. In order to not break *freshness* there must a requirement for when it is allowed to use both queries to the different parties. If an adversary ask for the long term key and the ephemeral

key to the same player it will conclusively obtain the same information as the player and thus break freshness and consequently not be able not use a **Test** query. Our requirement is that an adversary $\mathcal{A}$ can't ask both queries to the same player. Even with this requirement it is still necessary to revise the previous k now that $\mathcal{A}$ can obtain ephemeral keys. In the NAXOS protocol the intuition was to design a protocol that combined the long-term private key with the ephemeral key inside the hash function [4]. Therefore, an approach is to further add a value to the hash function as follows

$$k = H(Id_A, Id_B, X, Y, g^{ay}, g^{bx}, g^{xy}, g^{ab})$$

Our goal is to prove the three cases to be secure with the new key k, and it's done similarly to the way we proved it in our previous section. We begin by analysing the case where we have an initiator oracle without matching conversation. We know that we have a tuple (u, v, W) where W is the answer to the DH-problem. Our task is to let U and V be values in our protocol that we have control over and then the Gap Oracle will do the rest. In our first case, we control B and X, and therefor let $U = B$ and $V = X$. Our key k contains the keys $g^{ab}, g^{ay}, g^{bx}, g^{xy}$, because the adversary has access to both A and Y the keys g^{ab}, g^{ay} and g^{xy} are ruled out and we're left with g^{bx}, hence we let $U = B$ and $V = X$. Even though, we must take into account that the adversary can communicate with the responder beforehand and obtain information about X. Let X_e and Y_e be the ephemeral keys to adversary and the responder oracle respectively, and they both output key k. The adversary can use a **Reveal** query on the key k and our task is to know when it sends a query on the keys $g^{eb}, g^{ey_e}, g^{bx_e}, g^{y_e x_e}$. We let our assumption be that we have produced the keys, and from this assumption know g^{eb}, g^{ey_e} and $g^{y_e x_e}$. Considering no knowledge of g^{bx_e}, we must observe when the adversary ask for the hash-function. We repeatedly use the Gap Oracle whenever the adversary ask for the hash-function and verify if it is correct. Consider that this step doesn't give us W, however we ensure consistency of keys which ensures that the adversary works as usual.

Our second case is with a responder oracle without matching conversation. This case is similar to the one above, however we let U and V be other known values. Since we have control over A and Y we let $U = A$ and $V = y$ and run a similar approach as before.

The last case is easier to prove, because we have matching conversation and control both X and Y, hence we let $U = X$ and $V = Y$. In this case we're certain that X and Y won't be used in other conversations, even though there is a possibility that X is sent to other oracles we've already made the assumption and guessed that the oracle that will receive

this X and omit other cases. If we use Decisional Diffie-Hellman we obtain our W and use this value in our hash-function.

5.2 HMQV

The *Hash MQV* or simply *HMQV* protocol is a variant of MQV, where the MQV protocol is an efficient authenticated Diffie-Hellman protocol that uses public-key authentication. The HMQV protocol holds the MQV's security goals in the random oracle model under the computational Diffie-Hellman assumption [5]. Its communication is identical to the basic DH protocol as shown in Figure 2.2 except that the identities A, B may include a public-key certificate. Party A possesses a long-term private key a and corresponding public key $A = g^a$ and B's public and private key pair is $(b, B = g^b)$. The ephemeral DH values exchanged are $X = g^x$, $Y = g^y$, where x, y are chosen by the players A and B respectively. The protocol is depicted in Figure 5.1.

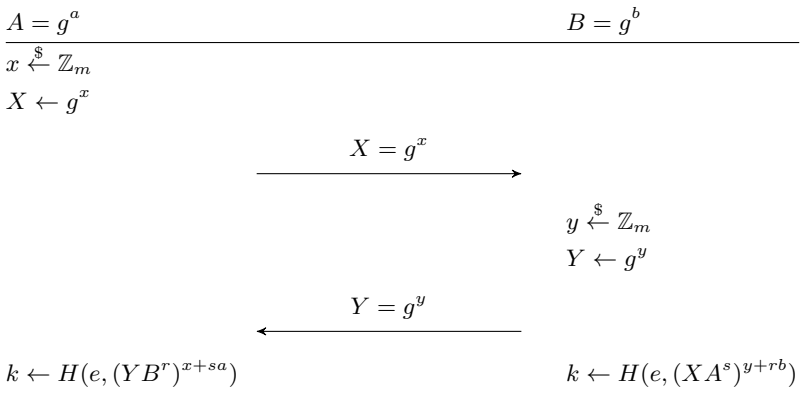


Figure 5.1: The HMQV protocol.

Where $e = (Id_A, Id_B, A, B, X, Y)$, $r = H(e, 2)$ and $s = H(e, 1)$.

The new key k still contain all elements that we introduced in chapter 5 only raised to the powers s, r and sr. If we consider $(XA^s)^{y+b}$ it is evident that we obtain $(g^{x+as})^{y+rb}$ which is $(g^{xy+say+rbx+srab})$. The last expression can be written as $g^{xy}(g^{ay})^s(g^{bx})^r(g^{ab})^{sr}$, that are the same elements that are contained in our previous calculations of k. If the key k is calculated differently however contain more or less the same values, why do we wish to use HMQV? The argument lies behind the number of exponentiations, the HMQV has a

fewer number of exponentiations than our previous protocol and is thereby much quicker to calculate. In addition, take notice that the values s and r are hash-values that are more or less random that implies in a way that we have a random linear combination. Because the values are random, it proves to be a difficult task for someone to predict what the linear combinations are unless you know all of the values. Accordingly, if an adversary is able to predict the keys k with random s and r it implies that the adversary is capable to predict all values. Our theory is that the security of HMQV should come trivially from the security of the last protocol.

We let $t = (g^{xy}(g^{ay})^s(g^{bx})^r(g^{ab})^{sr})$ and $t' = (g^{xy}(g^{ay})^{s'}(g^{bx})^{r'}(g^{ab})^{sr'})$. Where t is the calculation with a random s, r and t' is a new calculation with new s', r' when rewinding the protocol. If the adversary is capable to calculate both t and t' then it is able to obtain both r and s which is a trivial proof. However, if the adversary has knowledge of two values, say a public key and a ephemeral key, it is able to calculate t. We look at our previous cases where we put U and V for other values. In the case where we have a matching conversation, we let $U = X$ and $V = Y$. We know both of the values A and B and are capable of calculating g^{ab}, g^{ay} and g^{bx} and can easily obtain the value g^{xy}. However in the case where we have a responder, we let $U = A$ and $V = Y$ and we don't know the values A or Y. In this case we know the values g^{ab} and g^{bx} but not the values of g^{xy} and g^{ay}. We are interested in g^{ay} because it's the solution to the DH-problem. In this case we can, in the HMQV protocol, calculate $(\frac{t}{t'})^{(s-s')}$ and obtain g^{ay}.

To explain more detailed, we assume that the adversary manage to guess the value t correctly. We can test this by sending the values $(g^{x+as})^y$ and ax^s to the Gap Oracle, because we know the values (since we've chosen the values y). Then we rewind back and initiate a new run of the protocol, and if the adversary run the game as normal it will send X and receive Y back. At that point, when the adversary wants to calculate r and s we simply pick new values r' and s'.

Chapter 6

Concluding Remarks

We conclude this thesis by summarizing the desired goals that have been achieved in this project as well as acknowledging a few concluding remarks that can be helpful for further work. In this master thesis we essentially analyzed the security of key exchange protocols that use standard Diffie-Hellman communication, where our goal was to prove the secrecy of keys and authentication between the players. The initiative to use signed Diffie-Hellman was to provide authentication between the players against an active adversary. Still, this protocol can be broken if an adversary reveals the ephemeral key of the parties. We went about solving this problem by mainly adjusting a few steps in this protocol. Such a 2-move protocol did however achieve *implicit authentication* although with some modifications it was able to obtain *explicit authentication* with signatures. The KEA+ protocol does provide authentication between the players as well as secrecy of keys, even though it is a 2-move protocol, however it falls short when the adversary has the ability to ask for the ephemeral or long-term keys. Thus, our approach to update the calculation of the key k in the KEA+ protocol was enough to make this protocol secure against such adversary queries.

There exists various methods and ways to achieve a goal or proof. The path one chooses contains some but not all characteristics, and hence neglect other important elements. One aspect that was not discussed in this thesis is the fact the players in the signed Diffie-Hellman protocol are however left with an evidence that there has been a key exchange, namely the signatures. In the other protocols that were analyzed, the players don't have this kind of evidence.

Bibliography

[1] Mihir Bellare and Phillip Rogaway. *Entity authentication and key distribution*. 1993.

[2] Colin Boyd and Anish Mathuria. *Protocols for authentication and key establishment*. Springer Science & Business Media, 2013.

[3] Ran Canetti and Hugo Krawczyk. *Analysis of key-exchange protocols and their use for building secure channels*. 2001.

[4] Cas JF Cremers. Session-state reveal is stronger than ephemeral key reveal: Attacking the naxos authenticated key exchange protocol. In *International Conference on Applied Cryptography and Network Security*, pages 20–33. Springer, 2009.

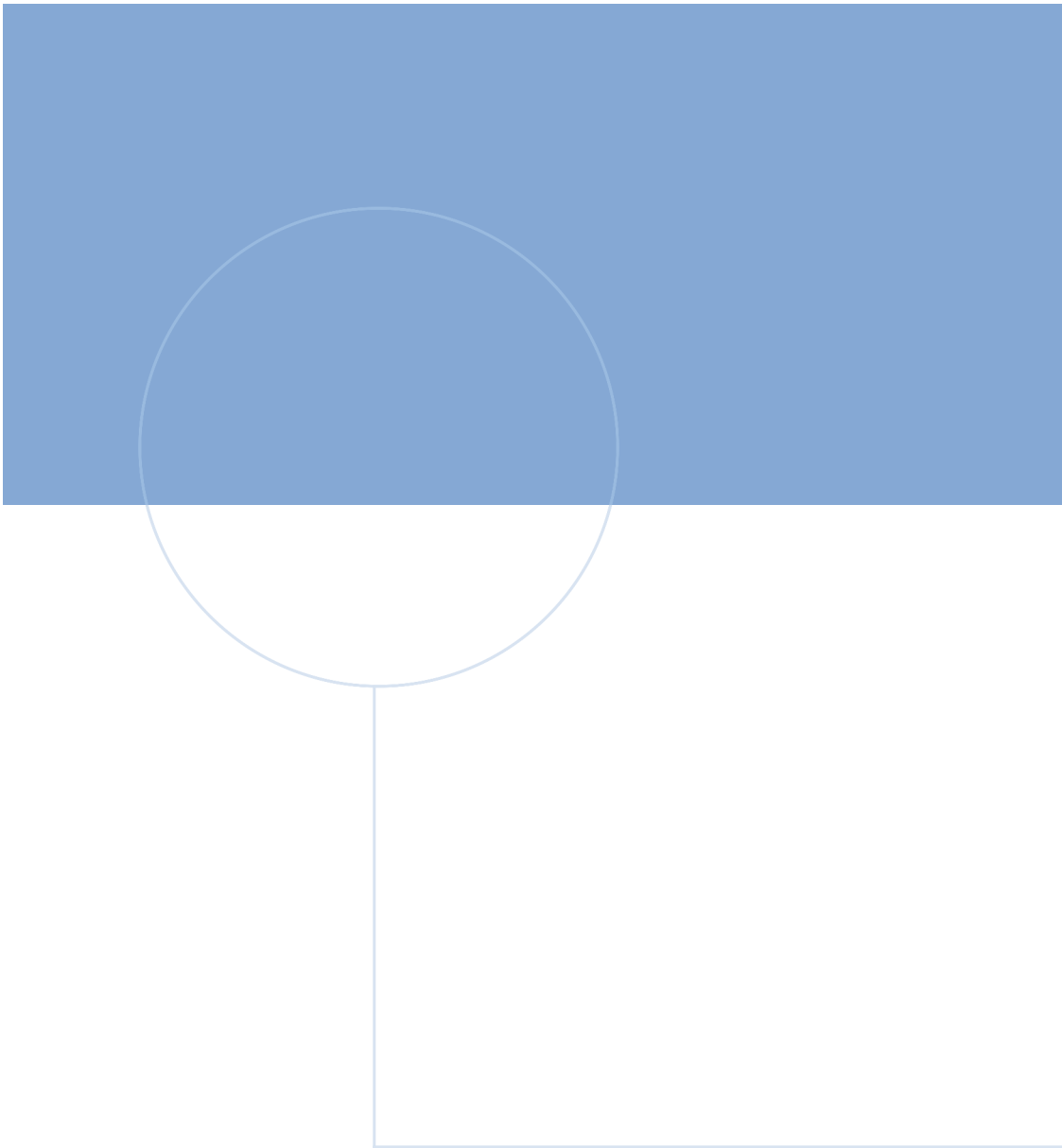
[5] Hugo Krawczyk. *HMQV: A high-performance secure Diffie-Hellman protocol*. 2005.

[6] Kristin Lauter and Anton Mityagin. *Security analysis of KEA authenticated key exchange protocol*. 2006.

[7] Tatsuaki Okamoto and David Pointcheval. The gap-problems: A new class of problems for the security of cryptographic schemes. In *International workshop on public key cryptography*, pages 104–118. Springer, 2001.

[8] Victor Shoup. *On formal models for secure key exchange*. Citeseer, 1999.

[9] William Stallings. *Cryptography and network security: principles and practice*. Pearson Upper Saddle River, 2017.



NTNU
Norwegian University of
Science and Technology