Security Analysis and Implementation of the Encrypted Key Exchange Protocol

Cryptography and Security Protocols

Filipe Viseu* and Mariana Costa†

June 2025

Contents

*ist1107091
†ist1107094

1 Introduction

A fundamental goal of cryptography is to enable secure communication over untrusted networks, even in the presence of active adversaries. To ensure both confidentiality and integrity, it is standard practice to apply encryption and authentication mechanisms. While public-key encryption and digital signatures provide a general and robust framework, their computational overhead can be prohibitive in resource-constrained environments or latency-sensitive applications.

An alternative approach is to first establish a shared secret key between the communicating parties using a key exchange protocol. This secret can then be used to derive keys for symmetric encryption and message authentication, offering a more efficient solution for securing subsequent communication.

A *Key Agreement* (KA) protocol enables two parties, say P and P', to agree on a common secret over a public channel. One of the main challenges faced by these protocols is resisting dictionary attacks, particularly *offline* ones: an attacker who intercepts protocol messages should not be able to efficiently test multiple password guesses without interacting with one of the honest parties, who can limitate the number of password guesses.

This problem is addressed by a class of protocols known as *Password-Authenticated Key Exchange* (PAKE). A PAKE protocol enables two parties, who share a low-entropy secret (typically a human-memorable password), to securely agree on a shared cryptographic key — even in the presence of an active adversary capable of intercepting or tampering with messages.

One of the earliest and most influential PAKE protocols is the *Encrypted Key Exchange* (EKE) protocol, introduced by Bellovin and Merritt in 1992 [BM92]. EKE follows the structure of a standard KA protocol but incorporates an additional encryption layer derived from the shared password to protect the key exchange messages from offline attacks.

Beyond its historical significance, the EKE protocol continues to influence the design of modern PAKE protocols. For instance, protocols such as SPAKE1 and SPAKE2 [AP05] can be viewed as specific instantiations of EKE, where concrete choices are made for the underlying public-key encryption scheme and key agreement mechanism. Similarly, more recent constructions like CPace [HL19] draw on core design principles introduced by EKE, demonstrating its lasting impact on the development of secure and efficient PAKE protocols.

Related Work

The original EKE paper, published in 1992 by Bellovin and Merritt [BM92], did not include a formal security proof. At the time, a rigorous security definition for PAKE protocols had not yet been developed. It was not until eight years later a formal game-based security model for PAKE was defined in [BPR00]. However, their model left several important questions unanswered, particularly concerning the exact security assumptions required for the underlying KA protocol (see, for instance, section 1.1 of [JRX24]).

A more robust definition, known as UC-security, was proposed five years later and has since become widely accepted as the standard. This definition is grounded in the Universal Composability (UC) framework, introduced by Canetti in [Can+05], which models security under arbitrary composition. This is especially relevant for PAKE protocols, where the negotiated session key is typically used in subsequent symmetric cryptographic schemes.

Our contributions

In this work, we survey the UC-security of EKE and its one-encryption variant, known as OEKE. We discuss the (Real-or-Random) RoR game-based security model and the proof, first presented in written form in [JRX24], that EKE with underlying plain Diffie-Hellman is not UC-secure. Furthermore, we provide a proof of the game-based security of EKE with underlying plain Diffie-Hellman, which we did not find in the consulted literature. We implement some of the attacks present in [JRX24], as well as the RSA-based protocol in [BM92]. All code can be found in this github repository[1]. We conclude by discussing the necessary conditions that the underlying KA protocol must satisfy for the EKE protocol to be UC-secure, as well as some inherent difficulties of studying these protocols.

[1] https://github.com/MarianaRioCosta/Encrypted-Key-Exchange

2 Preliminaries

In this section, we introduce the protocols we will be working with during the project.

Notation

We adopt the (standard) notation used in class and the notation from [JRX24].

2.1 Key Agreement

In this subsection, we introduce (Unauthenticated) Key Agreement Protocols.

Definition 1 (Key Agreement (KA) protocol, 2-round)**.** *A 2-round KA protocol consists of the following set of deterministic algorithms:*

- $\mathrm{msg}_1(a) = A \in \mathcal{M}_1$*, which is the message-generation function of party* P*;*
- $\mathrm{msg}_2(b, A) = B \in \mathcal{M}_2$*, which is the message-generation function of party* P'*;*
- $\mathrm{key}_1(a, B) = K \in \mathcal{K}$*, which is the key-computation function of party* P*;*
- $\mathrm{key}_2(b, A) = K' \in \mathcal{K}$*, which is the key-computation function of party* P'*.*

In a standard execution of the protocol:

1. Party P samples a random value $a \leftarrow \mathcal{R}$ and computes the first message $A := \mathsf{msg}_1(a)$, sending it to party P';
2. Upon receiving A, party P' samples $b \leftarrow \mathcal{R}$, computes $B := \mathsf{msg}_2(b, A)$, and sends it to P, while also computing the shared key $K' := \mathsf{key}_2(b, A)$;
3. Upon receiving B, party P computes the shared key $K := \mathsf{key}_1(a, B)$.

We also define correctness and security of these type of protocols. As expected, a KA scheme is said to be *correct* if both parties compute the same key, that is, $K = K'$.

Definition 2 (Correctness)**.** *A KA protocol is* correct *if*

$$\mathrm{key}_1\left(a, \mathrm{msg}_2\left(b, \mathrm{msg}_1(a)\right)\right) = \mathrm{key}_2\left(b, \mathrm{msg}_1(a)\right)$$

with overwhelming probability when $a, b \leftarrow \mathcal{R}$*.*

Definition 3 (Security)**.** *A KA protocol is secure if the following two distributions are indistinguishable:*

$a \leftarrow \mathcal{R}$	$a \leftarrow \mathcal{R}$
$b \leftarrow \mathcal{R}$	$b \leftarrow \mathcal{R}$
$A := \mathrm{msg}_1(a)$	$A := \mathrm{msg}_1(a)$
$B := \mathrm{msg}_2(b, A)$	$B := \mathrm{msg}_2(b, A)$
$K := \mathrm{key}_2(b, A)$	$K \leftarrow \mathcal{K}$
output (A, B, K)	output (A, B, K)

2.2 Password-Authenticated Key Exchange

A Password-Authenticated Key Exchange (PAKE) protocol enables two parties to securely establish a shared cryptographic key using only a low-entropy password that they have agreed upon in advance.

2.2.1 The Encrypted Key Exchange (EKE) Protocol

The *Encrypted Key Exchange* (EKE), introduced by Bellovin and Merritt in 1992 [BM92], was the first PAKE protocol ever proposed.

It follows the structure of a typical KA protocol, but with an additional encryption layer based on the shared password. Assuming that both parties P and P' share a secret password pw, the message A is encrypted before being sent. Specifically, party P sends $\mathcal{E}(\mathsf{pw}, A)$ instead of A, using a encryption scheme $\mathcal{E}$. Upon receiving the encrypted message, party P' decrypts it using $\mathcal{D}(\mathsf{pw}, \cdot)$ to retrieve A and proceed with the protocol, see Figure 1.

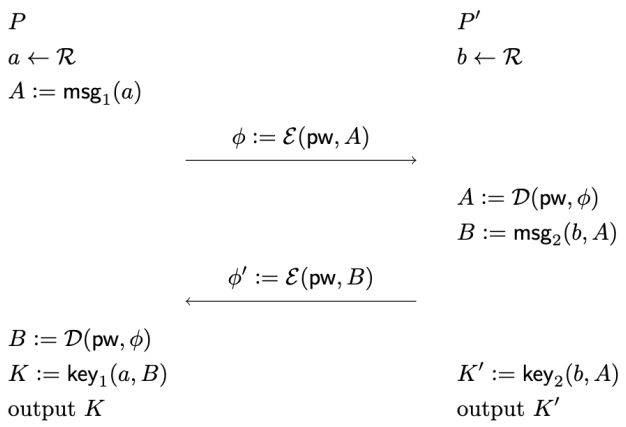

Figure 1: EKE with a 2-round KA protocol. This figure is taken from [JRX24].

This design prevents an eavesdropper from gaining useful information about the exchanged messages without knowledge of the password, thereby mitigating the risk of offline guessing attacks.

2.2.2 The One-encryption EKE (OEKE) protocol

The *One-Encryption EKE* (OEKE) protocol is one of the most prominent variants of the EKE protocol, first introduced in [BCP03] and illustrated in Figure 2.

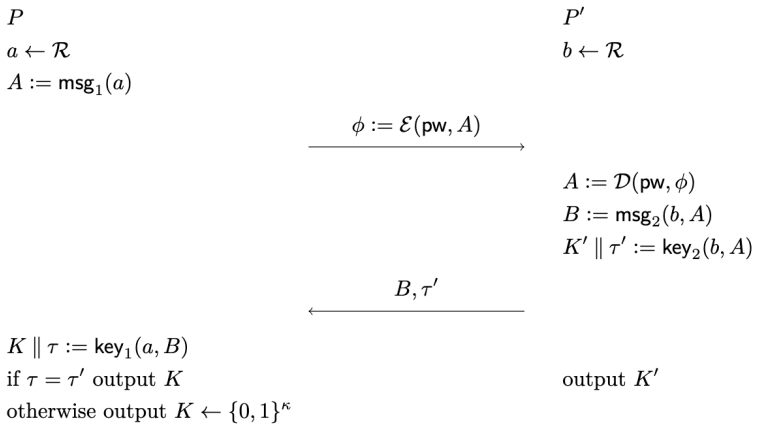

Figure 2: OEKE with a 2-round KA protocol. This figure is taken from [JRX24].

In this variant, only the message sent from P to P' is encrypted. Upon receiving it, P' replies with a plaintext key agreement message B and an authenticator τ, computed using its key agreement protocol key_2 as shown above. P' derives its session key from the first component of K'. Party P then processes

the KA message using its own key agreement protocol $\texttt{key}_1$, checks whether the received authenticator matches the second component of K, and, if so, accepts the first component of K as its session key.

3 Game-based Security

When designing cryptographic protocols, it's essential to analyze its security. Two standard game-based security models for PAKE protocols are the Real-or-Random (RoR) model and the Find-then-Guess (FtG) model [BPR00]. In the RoR model, a hidden bit b is chosen, and the adversary $\mathcal{A}$ can guess it across multiple sessions. In contrast, the FtG model allows guessing b in only one session. Thus, RoR provides a stronger security notion and we focus on this model. Following the expositions in [AP05], [Šal21] and [BPR00], we describe the model below.

As expected, in this model an adversary $\mathcal{A}$ interacts with protocol instances via the following oracles, representing possible real world capabilities of an adversary. Let $b \leftarrow \{0,1\}$ be a random bit and U_i denote an instance i of a given party, which can be either a server or client.

- $\texttt{Execute}(U_i, U_j)$: The adversary receives a complete honest session between the two parties U_i and U_j (passive network adversary).
- $\texttt{Send}(U_i, m)$: The adversary receives the message that the user U_i would generate upon receipt of message m (active network attack, in which the adversary may tamper with the message being sent).
- $\texttt{Corrupt}(U_i, \texttt{pw})$: The adversary learns user U_i password and all instance states of this user (models scenarios like password theft, Trojan installation or system compromise). The second argument allows altering its password by $\texttt{pw}$ (modeling attacks where a malicious client tampers with server-stored passwords).
- $\texttt{Reveal}(U_i)$ (not allowed if `Test` was queried on the same session) The adversary receives the final session key (representing a leakage from the subsequent use of the session key).
- $\texttt{Test}(U_i)$: (not allowed if `Reveal` was queried on the same session) The adversary is being given either the session key or a random value (depending on b) and must distinguish.

To prevent trivial attacks, we define a notion of *freshness*, allowing the command `Test` to be applied only on instances not trivially compromised. In this case, we say that the session is `fresh`.

Definition 4 (A game-based Security for PAKE)**.** *Consider a game that is initialized by sampling a password* pw *from the dictionary* Dict*, and by providing the PPT adversary $\mathcal{A}$ access to the five oracles. At the end of the game, $\mathcal{A}$ outputs its guess b' for the bit b used in the* Test*-query. The protocol is said to be secure if any PPT adversary $\mathcal{A}$ can win this game with only negligible advantage.*

4 EKE Realizes the Game-based Security Model

In [JRX24], the authors state that the game-based security of EKE is proved in [BPR00]. However, this document is just a long abstract, in which it is stated that the proof is in the extended version. We did not find this version online. Therefore, we tried to develop a proof by ourselves.

For simplicity, we use Diffie-Hellman as the underlying KA protocol.

Theorem 1. *Let EKE-DH be the Encrypted Key Exchange protocol using Diffie-Hellman over a cyclic group G of prime order p. In the Random Oracle Model, if the Decisional Diffie-Hellman (DDH) assumption holds in G, then EKE-DH is a secure PAKE protocol, achieving key indistinguishability against an active adversary (according to Definition 4).*

Proof. We prove the theorem by reduction. Assume the statement is false, i.e., there exists a PPT adversary $\mathcal{A}$ that breaks the IND-PAKE security of the EKE-DH protocol with a non-negligible advantage ε. We will construct a PPT algorithm $\mathcal{B}$ that uses $\mathcal{A}$ as a subroutine to solve the DDH problem in G with a non-negligible advantage.

First, we recall the definition of the protocol. Let parties be identified as $\{U_i\}_{i\in\mathbb{N}}$ and $\texttt{pw}_{ij}$ be the shared password, from a dictionary $\texttt{Dict}$, between the communicating parties (U_i, U_j). The protocol uses a symmetric encryption scheme $\Pi = (\text{Enc}, \text{Dec})$ and works as follows:

1. Alice chooses a random $x \in \mathbb{Z}_p$, computes $X = g^x$, and sends $c_A = \text{Enc}(\mathtt{pw}_{AB}, X)$ to Bob.

2. Bob receives c_A. He computes $X' = \text{Dec}(\mathtt{pw}_{AB}, c_A)$. If decryption succeeds, he recovers X. Bob chooses $y \in \mathbb{Z}_p$, computes $Y = g^y$, and sends $c_B = \text{Enc}(\mathtt{pw}_{AB}, Y)$ to Alice. Bob computes the session key $K = (X)^y = g^{xy}$.

3. Alice receives c_B, recovers $Y = \text{Dec}(\mathtt{pw}_{AB}, c_B)$, and computes $K = (Y)^x = g^{xy}$.

The adversary $\mathcal{A}$ interacts with a challenger via a set of oracles, and wins if it correctly guesses the bit b used in the $\mathcal{O}_{\mathtt{Test}}$ query. Since it can guess b at random with probability $\frac{1}{2}$, its advantage is given by

$$\mathbf{Adv}_{\mathcal{A}}^{\text{PAKE}} = \left| \Pr[\mathcal{A} \text{ wins}] - \frac{1}{2} \right|.$$

We now construct the DDH Distinguisher $\mathcal{B}$, which works as follows. The algorithm $\mathcal{B}$ receives as input a DDH challenge tuple (g, A, B, Z) where $A = g^a, B = g^b$ for random $a, b \in \mathbb{Z}_p$, and Z is either g^{ab} (a real DH tuple) or g^c for a random $c \in \mathbb{Z}_p$, and $\mathcal{B}$'s goal is to distinguish which case it is.

The algorithm $\mathcal{B}$ runs the adversary $\mathcal{A}$ and simulates its entire environment, selects two challenge identities, Alice* and Bob*, whose session will embed the DDH challenge and $\mathcal{B}$ does *not* know their password $\mathtt{pw}^* = \mathtt{pw}_{\text{Alice}^*,\text{Bob}^*}$. For any other pair of users (U_i, U_j), $\mathcal{B}$ generates and stores their password $\mathtt{pw}_{ij}$.

To simulate the encryption scheme Π, $\mathcal{B}$ maintains a list L_Π of tuples of the form $(\mathtt{pw}, M, c)$, meaning that $\text{Enc}(\mathtt{pw}, M) = c$. This allows $\mathcal{B}$ to simulate the oracles for $\mathcal{A}$. Since $\mathcal{A}$ does not know the password, real plaintexts are indistinguishable from random strings.

$\mathcal{B}$ responds to $\mathcal{A}$'s oracle queries as follows:

- $\mathcal{A}$ **calls** $\mathcal{O}_{\mathbf{Enc}}(\mathtt{pw}, M)$**:** When $\mathcal{A}$ queries the encryption oracle:
  - $\mathcal{B}$ checks its list L_Π for an entry $(\mathtt{pw}, M, \cdot)$. If a tuple $(\mathtt{pw}, M, c)$ exists, $\mathcal{B}$ returns c.
  - Otherwise, $\mathcal{B}$ generates a new, unique random string c' using Enc, adds the tuple $(\mathtt{pw}, M, c')$ to L_Π, and returns c' to $\mathcal{A}$.
- $\mathcal{A}$ **calls** $\mathcal{O}_{\mathbf{Dec}}(\mathtt{pw}, c)$**:** When $\mathcal{A}$ queries the decryption oracle:
  - $\mathcal{B}$ searches its list L_Π for an entry of the form $(\mathtt{pw}, \cdot, c)$.
  - If a tuple $(\mathtt{pw}, M, c)$ is found, $\mathcal{B}$ returns the plaintext M.
  - Otherwise, $\mathcal{B}$ returns an error symbol $\perp$, indicating decryption failure.
- $\mathcal{A}$ **calls** $\mathcal{O}_{\mathtt{Execute}}(U_i, U_j)$**:** This oracle simulates a complete, honest run of the protocol between users U_i and U_j, and returns the transcript to the adversary $\mathcal{A}$.
  - **Case 1: The session is between the challenge parties** $(\mathbf{Alice}^*, \mathbf{Bob}^*)$**.**
    1. $\mathcal{B}$ sets the ephemeral public key for Alice* to be A and for Bob* to be B from its DDH challenge.
    2. $\mathcal{B}$ invokes its own simulated encryption oracle to get the ciphertexts $c_A = \mathcal{O}_{\text{Enc}}(\mathtt{pw}^*, A)$ and $c_B = \mathcal{O}_{\text{Enc}}(\mathtt{pw}^*, B)$.
    3. $\mathcal{B}$ returns the transcript (c_A, c_B) to $\mathcal{A}$. The session key for this session, identified by sid^*, is implicitly set to be Z.
  - **Case 2: The session is between any other pair** (U_i, U_j)**.** In this case, $\mathcal{B}$ has full knowledge and control. It must perform a *real*, honest execution of the protocol.
    1. Since (U_i, U_j) are not the challenge pair, $\mathcal{B}$ knows their shared password $\mathtt{pw}_{ij}$.
    2. $\mathcal{B}$ generates fresh, random exponents for this session: $x_{ij}, y_{ij} \in \mathbb{Z}_p$.
    3. $\mathcal{B}$ computes the corresponding public keys $X_{ij} = g^{x_{ij}}$ and $Y_{ij} = g^{y_{ij}}$.
    4. Using the *real* encryption algorithm Enc and the known password $\mathtt{pw}_{ij}$, $\mathcal{B}$ computes the ciphertexts: $c_i = \text{Enc}(\mathtt{pw}_{ij}, X_{ij})$ and $c_j = \text{Enc}(\mathtt{pw}_{ij}, Y_{ij})$.
    5. $\mathcal{B}$ stores the session state, including the exponents and the resulting session key $K_{ij} = g^{x_{ij} y_{ij}}$, so it can answer future 'Reveal' queries for this session.

6. $\mathcal{B}$ returns the correctly computed transcript (c_i, c_j) to $\mathcal{A}$.

This simulation is computationally indistinguishable from a real-world execution, as $\mathcal{B}$ follows the protocol specification exactly for all non-challenge parties.

- **$\mathcal{A}$ calls $\mathcal{O}_{\texttt{Send}}(U_i, c)$:** This oracle models an active adversary sending a ciphertext c to a user U_i. $\mathcal{B}$'s response depends on the identity of U_i. Let's assume the message is from user U_j.
  - **Case 1: U_i is a challenge party (e.g., Alice*).** In this case, $\mathcal{B}$ does not know the password $\texttt{pw}^*$. To simulate Alice*'s behavior, $\mathcal{B}$ must use its simulated decryption oracle. It queries $\mathcal{O}_{\text{Dec}}(\texttt{pw}^*, c)$ to get a plaintext M.
    - If the result is $\perp$ (decryption failure), Alice* would reject the message. $\mathcal{B}$ returns no response or an error, perfectly mimicking the real protocol.
    - If the decryption succeeds and returns a valid group element $M = Y'$, $\mathcal{B}$ proceeds with the protocol for Alice*, using its challenge value A as its public key. It would compute a session key based on Y' and A. The simulation remains consistent from $\mathcal{A}$'s perspective.
  - **Case 2: U_i is not a challenge party.** In this case, $\mathcal{B}$ has full knowledge and control over U_i. Since $\mathcal{B}$ generated the passwords for all non-challenge pairs, it knows the password $\texttt{pw}_{ij}$ that U_i shares with U_j. Therefore, $\mathcal{B}$ can perform a *real*, not simulated, protocol action.
    - $\mathcal{B}$ computes $M \leftarrow \text{Dec}(\texttt{pw}_{ij}, c)$ using the actual decryption algorithm and the known password.
    - If decryption fails, $\mathcal{B}$ simulates U_i's abort/error behavior.
    - If decryption succeeds, $\mathcal{B}$ follows the EKE-DH protocol specification for U_i exactly. It uses U_i's internal state (which $\mathcal{B}$ maintains), generates any necessary ephemeral secrets, computes session keys, and generates the correct response ciphertext $c' = \text{Enc}(\texttt{pw}_{ij}, M')$. This response is then given to $\mathcal{A}$.

  Because $\mathcal{B}$ acts as an honest participant for all non-challenge parties, the simulation for these interactions is perfect and computationally indistinguishable from the real protocol.

- **$\mathcal{A}$ calls $\mathcal{O}_{\texttt{Corrupt}}(U_i)$:** If U_i is a challenge party, $\mathcal{B}$ aborts. Otherwise, it returns the stored password for U_i.

- **$\mathcal{A}$ calls $\mathcal{O}_{\texttt{Reveal}}(sid)$:** If $sid = sid^*$, $\mathcal{B}$ aborts. Otherwise, $\mathcal{B}$ can compute the session key as it knows the secrets for all non-challenge sessions, and returns the key.

- **$\mathcal{A}$ calls $\mathcal{O}_{\texttt{Test}}(sid^*)$:** We assume $\mathcal{A}$ calls this query on the challenge session sid^*.
  1. $\mathcal{B}$ returns its challenge value Z to $\mathcal{A}$ as the key.
  2. $\mathcal{A}$ completes its execution and outputs a guess $b' \in \{0, 1\}$.
  3. If $\mathcal{A}$ outputs $b' = 1$, $\mathcal{B}$ outputs 1 (guessing its DDH tuple was real).
  4. If $\mathcal{A}$ outputs $b' = 0$, $\mathcal{B}$ outputs 0 (guessing its DDH tuple was random).

We now analyze the probability that our constructed algorithm $\mathcal{B}$ succeeds in its DDH distinguishing game. The core of the reduction lies in the fact that $\mathcal{B}$'s simulation perfectly maps the DDH challenge to the IND-PAKE challenge presented to $\mathcal{A}$.

Let $\beta \in \{0, 1\}$ be the bit indicating if $\mathcal{B}$'s input tuple is real ($\beta = 1$) or random ($\beta = 0$) and $b \in \{0, 1\}$ be the bit for the challenger's key in $\mathcal{A}$'s PAKE game. Our simulation is constructed such that:

- If $\beta = 1$ (real DDH tuple), then $\mathcal{A}$ is presented with the real session key, which is a perfect simulation of the PAKE game with $b = 1$.

- If $\beta = 0$ (random DDH tuple), then $\mathcal{A}$ is presented with a random key, which is a perfect simulation of the PAKE game with $b = 0$.

Thus, the simulation ensures that the condition $\beta = b$ always holds. Furthermore, $\mathcal{B}$ determines its output guess, β', directly from $\mathcal{A}$'s output guess, b'. Specifically, $\beta' = b'$. Therefore, the event that $\mathcal{B}$ wins its game ($\beta' = \beta$) is identical to the event that $\mathcal{A}$ wins its game ($b' = b$). This gives us

$$\Pr[\mathcal{B} \text{ wins its game}] = \Pr[\mathcal{A} \text{ wins its game}].$$

and, by the definition of advantage in the IND-PAKE game, we know that

$$\Pr[\mathcal{A} \text{ wins its game}] \leq \frac{1}{2} + \mathbf{Adv}_{\mathcal{A}}^{\text{PAKE}}.$$

Combining these, the success probability of our distinguisher $\mathcal{B}$ is directly related to the advantage of $\mathcal{A}$. However, the standard advantage for a distinguishing problem like DDH is defined as the distinguishing gap,

$$\mathbf{Adv}_{\mathcal{B}}^{\text{DDH}} = |\Pr[\mathcal{B}(\text{real}) = 1] - \Pr[\mathcal{B}(\text{random}) = 1]|.$$

This gap is precisely twice the advantage of $\mathcal{A}$:

$$\mathbf{Adv}_{\mathcal{B}}^{\text{DDH}} = 2 \cdot \mathbf{Adv}_{\mathcal{A}}^{\text{PAKE}}.$$

This analysis holds conditional on the event $\neg\texttt{Guess}$, where `Guess` is the event that $\mathcal{A}$ successfully guesses the challenge password $\texttt{pw}^*$. Clearly, the probability of the `Guess` event is bounded by

$$\Pr[\texttt{Guess}] \leq \frac{q_\Pi}{|\mathcal{D}|},$$

where q_Π is the number of queries to the encryption/decryption oracles. Taking this bound into account, we can bound the advantage of our constructed algorithm $\mathcal{B}$ by

$$\mathbf{Adv}_{\mathcal{B}}^{\text{DDH}} \geq 2 \cdot (\mathbf{Adv}_{\mathcal{A}}^{\text{PAKE}} - \Pr[\texttt{Guess}]).$$

Since we assumed $\mathbf{Adv}_{\mathcal{A}}^{\text{PAKE}}$ is non-negligible and $\Pr[\texttt{Guess}]$ is negligible for a sufficiently large password dictionary, it follows that $\mathbf{Adv}_{\mathcal{B}}^{\text{DDH}}$ must be non-negligible. This contradicts the Decisional Diffie-Hellman assumption, which states that no PPT algorithm can have a non-negligible advantage in solving the DDH problem, and finishes the proof.

□

5 Universal Composability for PAKE Protocols

The Universal Composability (UC) framework was introduced by Canetti in [Can+05] and is a framework for proving the security of cryptographic protocols. Unlike traditional game-based models, such as those in the previous section, which analyze protocols in isolation, the UC framework ensures that a protocol remains secure even when composed with other arbitrary protocols. We reccommend [Can+05] for details.

A PAKE protocol is considered UC-secure if it *imitates* an ideal functionality denoted by $\mathcal{F}_{\text{PAKE}}$, meaning that the real-world protocol can emulate this ideal functionality indistinguishably and therefore, that the real protocol provides the same security guarantees as this ideal model. In figure 3 we illustrate the standard UC PAKE functionality.

- On input $(\mathsf{NewSession}, \mathsf{sid}, P, P', \mathsf{pw}, \mathsf{role})$ from P, send $(\mathsf{NewSession}, \mathsf{sid}, P, P', \mathsf{role})$ to $\mathcal{S}$. Furthermore, if this is the first NewSession message for *sid*, or this is the second NewSession message for sid and there is a record $\langle P', P, \cdot \rangle$, then record $\langle P, P', \mathsf{pw} \rangle$ and mark it fresh.
- On $(\mathsf{TestPwd}, \mathsf{sid}, P, \mathsf{pw}^*)$ from $\mathcal{S}$, if there is a record $\langle P, P', \mathsf{pw} \rangle$ marked fresh, then do:
  - If $\mathsf{pw}^* = \mathsf{pw}$, then mark the record compromised and send "correct guess" to $\mathcal{S}$.
  - If $\mathsf{pw}^* \neq \mathsf{pw}$, then mark the record interrupted and send "wrong guess" to $\mathcal{S}$.
- On $(\mathsf{NewKey}, \mathsf{sid}, P, K^* \in \{0,1\}^\kappa)$ from $\mathcal{S}$, if there is a record $\langle P, P', \mathsf{pw} \rangle$, and this is the first NewKey message for *sid* and P, then output (sid, K) to P, where K is defined as follows:
  - If the record is compromised, then set $K := K^*$.
  - If the record is fresh, a key (sid, K') has been output to P', at which time there was a record $\langle P', P, \mathsf{pw} \rangle$ marked fresh, then set $K := K'$.
  - Otherwise sample $K \leftarrow \{0,1\}^\kappa$.

  Finally, mark the record completed.

Figure 3: UC PAKE functionality $\mathcal{F}_{\text{PAKE}}$. This figure is taken from [JRX24].

This functionality involves two parties, P with password `pw`, and P' with password `pw'`. Each protocol execution creates a session for P and a corresponding session for P'. Each session maintains a state, which evolves over time as follows:

- When a session is created using the `NewSession` command, it is marked as `fresh`. It remains in this state unless actively attacked by the (ideal) adversary.
- The adversary can attempt an online guessing attack on a fresh session by issuing a `TestPwd` command for P, along with a guessed password `pw*`. This simulates the adversary executing P''s algorithm with `pw*` and interacting with P. If the guess is correct (`pw* = pw`), the session becomes `compromised`; otherwise, it becomes `interrupted`. Similarly, the adversary may issue `TestPwd` for P'. Importantly, once a session is either `compromised` or `interrupted`, it can never revert to `fresh`, and `TestPwd` may only be used once per session.
- The adversary may terminate a session using the `NewKey` command. In response, the session outputs a key determined by its own state and the state of its peer session. The session is then marked as `completed`, and further `TestPwd` commands are disallowed.

The session key generation logic is nuanced and central to the security guarantees, and is outlined as follows:

- **Correct execution.** If both sessions (P and P') are `fresh`, and `pw = pw'`, the adversary has not interfered and the protocol has executed correctly. If P' ends first, it outputs a random session key `K'`. Later, when P ends, it outputs the same key $K = K'$.
- **Mismatched passwords.** If both sessions are `fresh`, but `pw` $\neq$ `pw'`, this models a benign mismatch. In this case, both P and P' output independent random session keys.
- **Successful online attack.** If a session is `compromised`, it reflects a successful password guess. Security is forfeited, and the adversary may choose the session key. This applies symmetrically to both P and P'.
- **Failed online attack.** If a session is `interrupted`, the password guess was incorrect. The adversary gains no information, and the session outputs an independent random key.
- **Only one side attacked.** If P is `fresh`, but P' is either `compromised` or `interrupted`, the key output by P must remain independent of any adversarial influence. Thus, P outputs a random session key.

In all cases where a session outputs a random key, the adversary learns nothing about it. Even if the adversary later obtains a party's password after the session completes, they cannot retroactively issue a `TestPwd` command. For instance, if P remains `fresh` while P' is `compromised`, the session key of P is still fully hidden from the adversary, despite their full control over P''s key.

In these terms, the UC framework defines a perfect version of what a secure PAKE protocol should achieve: namely, matching passwords result in a shared session key, while mismatches or attacks result in no information leakage beyond what the ideal model allows.

6 EKE with plain Diffie-Hellman is not UC-secure

In this section, we present the proof of the following theorem from [JRX24], which proves that EKE instantiated with plain Diffie-Hellman is not UC-secure.

Theorem 2. *The EKE protocol with underlying plain Diffie-Hellman does not UC-realize $\mathcal{F}_{PAKE}$.*

Proof. Our goal is to prove that the real protocol can be distinguished from its idealized version with non-negligible probability. Assume there exists a simulator that runs both protocols in an indistinguishable way. Furthermore, consider an adversary $\mathcal{A}$ who correctly guesses the password pw and disrupts the communication by changing the message g^b sent by P' to $g^{2b} = (g^b)^2$, as illustrated below:[2]

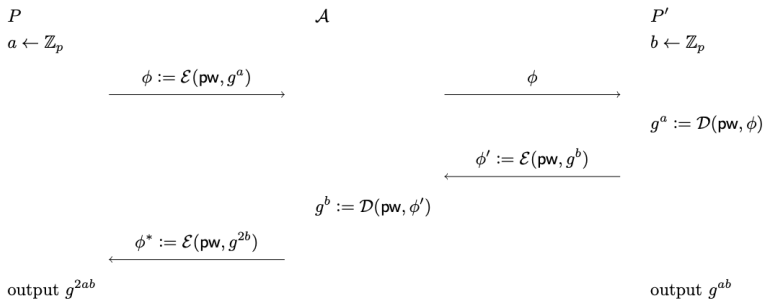

Figure 4: Interaction of $\mathcal{A}$ with P and P'. This figure is taken from [JRX24].

Now, consider the *environment* $\mathcal{Z}$ explained in Figure 5, which outputs 1 if the key K computed by P after the attack by $\mathcal{A}$ is $(K')^2$ and outputs 0 otherwise:

1. Sample $\mathsf{pw} \leftarrow \mathsf{Dict}$, and send $(\mathsf{NewSession}, \mathsf{sid}, P, P', \mathsf{pw})$ to P and $(\mathsf{NewSession}, \mathsf{sid}, P', P, \mathsf{pw})$ to P'. // let P and P' run the protocol on the same password pw
2. On ϕ from P, instruct the adversary to send ϕ to P'. Observe the output of P', K'. // pass the P-to-P' message without any modification
3. On ϕ' from P', query $B := \mathcal{D}(\mathsf{pw}, \phi')$ and then $\phi^* := \mathcal{E}(\mathsf{pw}, B^2)$, and instruct the adversary to send ϕ^* to P. Observe the output of P, K. // replace the P'-to-P message $\mathcal{E}(\mathsf{pw}, g^b)$ with $\mathcal{E}(\mathsf{pw}, g^{2b})$
4. Output 1 if $K = (K')^2$, and 0 otherwise. // guess "real world" iff $K = (K')^2$

Figure 5: Environment $\mathcal{Z}$. This figure is taken from [JRX24].

In the real world, since $\mathcal{A}$ successfully modifies the message g^b to g^{2b}, the resulting keys satisfy $K = (K')^2$ with probability 1. Thus, $\mathcal{Z}$ outputs 1 with probability 1 in the real-world scenario.

We now compute the probability that $\mathcal{Z}$ outputs 1 in the ideal world, separating the cases where the session is compromised before P' outputting the key K' and not compromised before P' outputting the key K'.

[2] See Section 7.1.2 and the GitHub repository for the computational implementation of this attack.

Define the event `Compromise` as the event that the P' session is compromised before it output its key. Looking at Figure 5, we see that this happens if and only if the simulator $\mathcal{S}$ correctly guesses the password $\mathtt{pw}' = \mathtt{pw}$. Since $\mathtt{pw}$ is chosen uniformly from the dictionary `Dict`, we conclude that

$$\Pr[\mathtt{Compromise}] \leq \frac{1}{|\mathtt{Dict}|}.$$

We now assume that `Compromise` does not occur, that is, the P' was not compromised before outputting K' and is still `fresh`. In this scenario, the state of P before outputting K can be one of the following: `fresh`, `interrupted`, or `compromised`, as explained in the previous section.

- In the `fresh` case, the session has not been attacked and we have $K = K'$ and $K = (K')^2$ if and only if $K = e$, which evidently occurs with probability $1/|G|$;
- In the `interrupted` case, the generated session key of P must be independent of K' and is generated randomly, revealing no extra information. The probability that $K = (K')^2$ is, again, $1/|G|$;
- In the `compromised` case, all security guarantees are lost and the adversary choose the session key for P. The probability that $K = (K')^2$ is $1/|G|$.

We deduce that the overall probability that $\mathcal{Z}$ outputs 1 in the ideal world is at most

$$\Pr_{\text{ideal}}[\mathcal{Z} = 1] \leq \frac{1}{|\mathtt{Dict}|} + \frac{1}{|G|},$$

Thus, the distinguishing advantage of $\mathcal{Z}$, i.e, the difference in probabilities that it outputs 1 in the real and ideal worlds is, at least,

$$1 - \frac{1}{|\mathtt{Dict}|} - \frac{1}{|G|}$$

which is non-negligible, since we assume $|\mathtt{Dict}| \geq 2$. This concludes the proof. □

7 Implementation

We implement a proof of concept for both attacks against plain Diffie-Hellman based EKE in [JRX24, Section 3.1, 3.2]. Furthermore, we provide an implementation of Bellovin and Merritt's EKE introduced in 1992 [BM92], using RSA. All code can be found in this github repository[3]. We included a video with a test run of all the scripts, `poc.mp4`.

7.1 Diffie Hellman Based EKE

We started by implementing a simple proof of concept of OEKE-PRF where the underlying KA protocol is plain Diffie-Hellman.

That is, Alice sends $\mathcal{E}(\mathtt{pw}, g^a)$ to Bob. Then, Bob samples an integer b, computes the key agreement key $K' = (g^a)^b = g^{ab}$, and derives the PAKE session key as $\mathsf{PRF}_{K'}(0)$. Next, Bob sends g^b together with $\tau' = \mathsf{PRF}_{K'}(1)$ to Alice. Alice then computes $K = g^{ab} = (g^b)^a$, and checks whether $\tau' = \mathsf{PRF}_K(1)$. If so, Alice outputs $\mathsf{PRF}_K(0)$ as its session key; otherwise, outputs a random session key.

7.1.1 Allowing Identity Element in Diffie-Hellman

Suppose two parties, P and P', are using the OEKE protocol (recall Figure 2) with plain Diffie-Hellman as the underlying KA protocol. That is, for some arbitrary generator g of a cyclic group, party P sends an encrypted message $\mathtt{msg}_1(a) = g^a$, and party P' responds with $\mathtt{msg}_1(b) = g^b$ (along with an authenticar τ'). Under these conditions, there is a critical flaw: an adversary can learn the session key of P by sending the identity element e of the group to party P. Then, P will compute $K = e^a = e$, and the adversary predicts its session key, thereby breaking the security of the OEKE schemes based on plain Diffie-Hellman.

[3] https://github.com/MarianaRioCosta/Encrypted-Key-Exchange

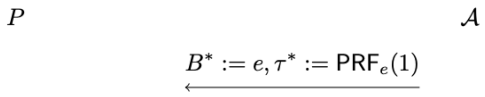

Figure 6: Attack on OEKE-PRF with plain Diffie-Hellman. $\mathcal{A}$ (that does not know `pw`) sends a single message to P and can predict the session key of P. This figure is taken from [JRX24].

Note that this attack does not apply directly to EKE, as P only receives encrypted messages. Consequently, the adversary would need to know the password `pw` in order to carry out the attack. For the same reason, this attack does not apply directly to party P' in the OEKE setting.

An obvious countermeasure to this attack is to restrict the set of group elements that party P accepts when computing the session key. For example, if the chosen cyclic group is $(\mathbb{Z}_p, +)$, where p is a prime, P should reject 0.

7.1.2 A Man-In-The-Middle Attack for plain Diffie-Hellman

We also implemented a proof of concept for the attack explained in Theorem 2. We simulate how an adversary $\mathcal{A}$ who correctly guesses the password `pw` can disrupt the communication by changing the message g^b sent by P' to $g^{2b} = (g^b)^2$.

7.2 RSA-based EKE

In `RSA` we provide an implementation of EKE, as proposed in [BM92]. We use RSA as the asymmetric cipher and AES-128-ECB as the symmetric cipher.

Implementing EKE in general requires a few considerations. Some of the subtleties of using RSA are recognized in [BM92]. We explain them here in more detail.

Firstly, encoding the RSA public key in a way that ensures indistinguishability from a random string is non-trivial. In particular, if both components of the key, (n, e), are encrypted using a password, an attacker attempting an offline dictionary attack could easily test candidate decryptions by checking whether the resulting n' has small prime factors. Since most random strings do not represent a product of two large primes, this allows the attacker to efficiently distinguish valid RSA moduli from random-looking data, thereby filtering out incorrect password guesses and significantly weakening the security of the scheme. Therefore, we must only encode e. We implemented the encoding proposed in [BM92]. That is, we begin with the binary encoding of e, and add 1 with probability 1/2. This addition is required because all possible values of e are odd: otherwise $\phi(n), e$ would not be coprime and decryption would not me unique.

In [BM92], the authors provide an intuition for why this encoding works, but we formalized it into a concrete proof.

Theorem 3. *Let $n = pq$, where p and q are large primes. Suppose e is sampled uniformly at random among odd integers modulo n such that $\gcd(e, \varphi(n)) = 1$. Then, with overwhelming probability, the proposed encoding of e (binary encoding plus addition of 1 with probability 1/2) is computationally indistinguishable from a random integer modulo n.*

Proof. Consider the choice of p and q as safe primes, i.e., of the form $p = 2p_0 + 1$ and $q = 2q_0 + 1$, where p_0 and q_0 are themselves prime. Then $\varphi(n) = (p-1)(q-1) = 4p_0q_0$, which implies that the prime factorization of $\varphi(n)$ is well-structured and consists mostly of large prime factors.

Because of this, an overwhelming majority of odd integers modulo n are coprime to $\varphi(n)$. More precisely, the proportion of odd integers modulo n coprime to $\varphi(n)$ is

$$\frac{\varphi(\varphi(n))}{\frac{1}{2}\varphi(n)} = \frac{2 \cdot 2 \cdot (p_0 - 1) \cdot (q_0 - 1)}{4p_0q_0} = \frac{(p_0 - 1) \cdot (q_0 - 1)}{p_0q_0} \approx 1.$$

If p, q have a different structure, in order to be safe primes they must still not be smooth, that is, they must not have all prime factors small, otherwise Pollard's rho algorithm could be used as a valid attack. Therefore, this step can be generalized for safe p and q.

Consequently, from the adversary's perspective, a randomly chosen odd integer modulo n is, with overwhelming probability, a valid candidate for e. This means that the encoding of e — a random odd integer with the appropriate length padding — is computationally indistinguishable from a uniformly random integer modulo n. Therefore, we conclude that an adversary attempting a dictionary attack on the encoded e obtains extremely little information, and cannot distinguish the encoding from a random integer modulo n with sufficient advantage.

□

The fact that n is transmitted in the clear introduces a subtle vulnerability in the protocol. In particular, an active adversary could replace the original modulus n with a different modulus n' in the first message, causing party B to receive the modified pair $\langle P(e), n' \rangle$. Consequently, B's response will be of the form

$$(R, \mathsf{challenge}_B)^e \bmod n'.$$

Now consider a potential dictionary attack. From a guessed password P_0, the adversary computes

$$e_0 = P_0^{-1}(P(e)).$$

Let R be the random secret key. Since the adversary chose n', they also know its factorization. This enables them to compute the corresponding private decryption exponent d' for e_0, assuming $\gcd(e_0, \phi(n')) = 1$. With this, they can decrypt B's ciphertext to obtain

$$(R, \mathsf{challenge}_B)^{ed'} \bmod n'.$$

If $e_0 \neq e$, the decryption will yield a pseudorandom value. If $e_0 = e$, the correct plaintext $(R, \mathsf{challenge}_B)$ is recovered. However, this does not help the adversary, as both R and $\mathsf{challenge}_B$ are uniformly random and unknown. To proceed with the protocol, the adversary would need to compute a valid response of the form

$$R(\mathsf{challenge}_A, \mathsf{challenge}_B),$$

but without knowing either R or $\mathsf{challenge}_B$, they cannot do so.

Therefore, although the adversary can manipulate the modulus and attempt decryption using guessed passwords, the decrypted output is either meaningless or unusable. This prevents them from confirming whether a password guess is correct, and the dictionary attack fails.

One further consequence of transmitting n in the clear is that it exposes the user to potential cryptanalysis. Specifically, if n is available to an attacker, it may be possible — given sufficient computational resources — to factor it. This, in turn, would allow the adversary to recover the random value R and potentially mount an attack on the password P. In contrast, if n were not known to the adversary, they would face a much more difficult task: attempting to analyze ciphertexts whose plaintexts are uniformly random. In such a scenario, the attacker is reduced to solving a system with no known structure in the plaintext space, which is considered computationally infeasible.

7.2.1 A Man-In-The-Middle Attack for RSA

We also implement a Man-In-The-Middle (MITM) between the server and the client Without the encoding specified above, this Man-In-The-Middle functionality could then be used to break the protocol. However, since our implementation is protected against this attack, for now our MITM only forwards the communication. We explain the partition attack in this subsection.

We have stated that encryptions using P must leak no information. However, ensuring this property is often challenging due to the numerical structure of the cryptosystems in use. For instance, in the RSA setting, public exponents e are always odd. If no special precautions are taken, an attacker could discard approximately half of the candidate passwords P_0 simply by observing whether $P_0^{-1}(P(e))$ is even.

While this reduction in the keyspace may initially appear harmless, it can in fact lead to a devastating vulnerability. Recall that each session generates a fresh public key e, independent of previous sessions. As a result, for each new session, trial decryptions using an incorrect guess P_0 will either yield a valid odd exponent e_0 or an invalid even one. Hence, every session effectively partitions the remaining set of candidate passwords into two nearly equal subsets.

This leads to partition attacks. With each observed session, the adversary halves the set of viable passwords. The result is a logarithmic decrease in the search space, meaning that relatively few intercepted sessions are sufficient to eliminate all incorrect guesses and recover the correct password P.

This shows the importance of the security mechanisms we implemented, which are essential to prevent such cumulative leakage and to ensure the long-term confidentiality of the shared secret.

8 Requirements on the underlying KA protocol in (O)EKE

There are some known requirements, presented in [JRX24], the underlying KA protocol needs to meet to be UC-secure. Below, we briefly give a overview of them and the intuition behind them. For details, we refer the reader to this reference.

8.1 Collision-Resistance

In the previous sections, we assumed that the (O)EKE protocol uses (some variants of) the Diffie-Hellman. However, the attack described in Section 7.1.1. shows that any KA protocol underlying OEKE must satisfy a form of contributory property. Party P must meaningfully influence the resulting key, and party P' must not be able to significantly bias its distribution on their own, as the attacker did by sending the identity group element to P. This provides the required intuition for collision resistance.

Definition 5 ([JRX24], Definition 3.1)**.** *A KA protocol is* collision-resistant *if the key space is $\mathcal{K} = \{0,1\}^{3\kappa}$, and for any polynomially bounded q and any PPT adversary $\mathcal{A}$, the winning probability of $\mathcal{A}$ in the following game is negligible:*

$$
\begin{aligned}
&\forall i \in [q] : a_i \leftarrow \mathcal{R} \\
&\forall i \in [q] : A_i := \mathrm{msg}_1(a_i) \\
&B^* \leftarrow \mathcal{A}(A_1, \ldots, A_q) \\
&\forall i \in [q] : K_i \| \tau_i := \mathrm{key}_1(a_i, B^*) \\
&\mathcal{A}\,\text{wins if } \exists i \neq j : \tau_i = \tau_j
\end{aligned}
$$

Here, we split keys $key_1(a_i, B^*) \in \{0,1\}^{3\kappa}$ into two chunks: $K \in \{0,1\}^{\kappa}$ and $\tau \in \{0,1\}^{2\kappa}$.

8.2 Strong Pseudorandomness

The security of EKE also relies on the underlying key agreement KA protocol satisfying a property known as *strong pseudorandomness*. This is not an issue for the Diffie–Hellman protocol, as its messages are uniformly distributed. However, in general, strong pseudorandomness does not automatically follow from standard security notions or even from basic pseudorandomness[4], which is strandard to assume, and must therefore be treated as an independent requirement.

In a KA protocol, the first message A, the second message B, and the shared key K can each be either *real* (i.e., honestly generated according to the protocol) or *random*. This gives rise to $2^3 = 8$ possible combinations for the tuple (A, B, K). One of these combinations — namely, the (random, random, real) case —clearly does not make sense. If both messages are random, then no party has followed the protocol, and there is no meaningful way to generate a real key K. Consequently, only the remaining seven combinations are considered here.

The standard security definition in Definition 3 guarantee that the distributions (real, real, random) and (real, real, real) are indistinguishable. Together with basic pseudorandomness, this implies indistinguishability among five distributions[5]. Additionally, Lemma 5.2 of [JRX24] can be used to show that the (random, real, real) and (real, real, real) are also indistinguishable. Altogether, standard security and basic pseudorandomness account for six indistinguishable distributions.

The remaining case is (real, random, real). Proving that this distribution is indistinguishable from (real, real, real) is challenging: we would need to produce a real key K without knowing either the secrets a or b, which are used to generate A and B, respectively. To overcome this, the authors require that the

[4] As shown by the counterexample on pages 20 and 21 of [JRX24].

[5] Pseudorandomness (see Section 2.1 of [JRX24]) ensures that the following four distributions are indistinguishable: (real, real, random), (real, random, random), (random, real, random), and (random, random, random).

distributions (real A, real B) and (real A, random B) be indistinguishable even when the shared key K is known. This stronger condition is referred to as *strong pseudorandomness*.

In summary, standard security combined with strong pseudorandomness ensures that all seven valid combinations of (A, B, K) are computationally indistinguishable. Without strong pseudorandomness, the (real, random, real) case may remain distinguishable, thereby imposing this additional requirement on the underlying KA protocol.

8.3 Pseudorandom Non-malleability

The MITM attacks we presented before point to the security property that the P'-to-P message and the key of P must both be pseudorandom, ensuring that, even a malicious attacker attempting to manipulate the second message during the protocol execution, cannot make the generated key of P predictable or dependent on the manipulated information.

Definition 6 ([JRX24], Definition 3.6)**.** *A KA protocol is pseudorandom non-malleable if the following two distributions are indistinguishable:*

$a \leftarrow \mathcal{R}$	$a \leftarrow \mathcal{R}$
$b \leftarrow \mathcal{R}$	
$A := \text{msg}_1(a)$	$A := \text{msg}_1(a)$
$B := \text{msg}_2(b, A)$	$B := \mathcal{M}_2$
$K' := \text{key}_2(b, A)$	$K' := \mathcal{K}$
$B^* \leftarrow \mathcal{A}(A, B, \mathcal{K}')$	$B^* \leftarrow \mathcal{A}(A, B, \mathcal{K}')$
abort if $B^* = B$	abort if $B^* = B$
$K := \text{key}_1(a, B^*)$	$K := \text{key}_1(a, B^*)$
output K to $\mathcal{A}$	output K to $\mathcal{A}$

Under pseudorandom non-malleability, such modification (in this case, squaring the message $B = g^b$) does not produce a predictable or correlated session key K. Instead, the distribution of K given B^* is statistically close to a uniformly random value, independent of the original message B.

9 Final Considerations

PAKE protocols were a topic with which both of us were unfamiliar when we chose this topic.

This project allowed us to relate some notions learnt in class with protocols we had never worked with before. It resulted in us reading 10+ papers and definitely expanded our horizons. Our biggest difficulties were getting familiar with all the different experiments and functionalities, as well as comprehending the security definitions. We expand more on this below. After reading some of the literature in this topic, as well as experimenting with the implementation and some attacks, there are a few remarks we would like to share.

In this section, we discuss the applicability of the attacks in [JRX24], the (lack of) simplicity of the security notions and some of the mistakes present in other literature.

How real are these attacks?

As discussed in [JRX24], the attacks outlined in Section 7.1 do not directly compromise the protocol in an obvious "real-world" scenario. However, these attacks may still pose a threat when the protocol is used as a component in a more complex system. In particular, [Han+25] proposes a weaker variant of the UC-PAKE functionality, known as lazy-extraction PAKE (lePAKE), and shows that several existing PAKE constructions satisfy this relaxed definition, even if they fail to meet the full UC-PAKE specification. While lePAKE may seem secure in isolation, it has been observed in [Sho20] that combining such a protocol with a secure channel can lead to critical vulnerabilities in the composed system.

The partition attack we explained in 7.2 initially seems like a more practical attack. However, if the protocol has a proper encoding of the public key, this is no longer the case. It is also worth noting that it would require $\mathcal{O}(\log_2(N))$ requests, becoming a possibly expensive attack. Bellovin and Merritt also provide some suggestion of strengthening EKE against cryptanalytic attacks [BM92, Section 2.5.3].

Our view on the security notions for PAKE

In both the UC and game-based security frameworks, the central goal is to ensure that an adversary is limited to a single online password guess per session. Security then relies on the assumption that the attacker must correctly guess the password to succeed — an event with non-negligible probability, yet unavoidable in any password-based setting. The objective is to guarantee that no other form of attack becomes feasible beyond this inherent vulnerability.

Our first reactions to the security definitions for PAKE was that these seemed pretty far from our intuition. In particular, UC-security was very different from anything we had previously worked with in class.

Even though we managed to prove by ourselves the game-based security of EKE with plain Diffie-Hellman, understanding the UC-security of the protocol became a much more difficult task. Fortunately, the main paper we studied [JRX24] gives a proof for a more efficient version of EKE, while other works [MRR20; Erw+20; SGJ23; Beg+23] provide proofs with some fundamental differences among them, and some of them even with flaws. An overview of these flawed analysis is stated in [JRX24, Table 2]. Although [JRX24] strives for a high level of generality, this comes at the cost of clarity, as the proof involves several independent subtleties that can make it difficult to follow. The tutorial version by Xu [Xu25] provided a more detailed explanation, while also commenting on the general feeling of lack of simplicity and intuition that make these definitions hard to understand for a first time reader.

Flaws in existing works

In the previous subsection we mentioned that some works present flaws in their proofs of security for EKE. We found it natural to wonder where these proofs go wrong. We now present the five lessons identified by [Xu25], which serve as general principles about security proofs in UC and/or for PAKE. The errors found in literature come from not respecting some of these.

1. For the security of PAKE, we must consider a man-in-the-middle adversary that may see and modify (or pass without modification) protocol messages in both directions.

2. Before writing down a UC proof, one must fully understand the ideal functionality. For example, in UC PAKE one must understand what the states `fresh`, `compromised` and `interrupted` mean, as well as how the session keys depend on them.

3. As we saw, game-based PAKE and UC PAKE are very different security notions; it is dangerous to give a game-based security proof and then say it "extends" to UC.

4. Before declaring two cases are similar and thus the second case does not need a detailed argument, one must thoroughly check that this is indeed true.

5. A reduction loses some information while simulating the security game to the adversary (due to embedding the challenge into parts of the security game), and one must go through the entire reduction and make sure that the "secret information" is not needed somewhere other than our main focus.

As the first and arguably most extensively studied PAKE protocol, EKE lacks a security proof that is both rigorous and clearly articulated—a need that, until recently, has remained largely overdue. Among the existing literature, the work of [JRX24] provides the most comprehensive and well-structured analysis, which underscores its significance as the primary reference for our study.

References

[AP05] Michel Abdalla and David Pointcheval. "Simple Password-Based Encrypted Key Exchange Protocols". In: *Topics in Cryptology – CT-RSA 2005*. Ed. by Alfred Menezes. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 191–208.

[Beg+23] Hugo Beguinet et al. "GeT a CAKE: Generic Transformations from Key Encapsulation Mechanisms to Password-Authenticated Key Exchanges". In: *Applied Cryptography and Network Security – ACNS 2023, Part II*. Ed. by Mehdi Tibouchi and Xiaofeng Wang. Vol. 13906. Lecture Notes in Computer Science. Springer, Cham, June 2023, pp. 516–538. DOI: 10.1007/978-3-031-33626-3_24.

[BPR00] Mihir Bellare, David Pointcheval, and Phillip Rogaway. "Authenticated key exchange secure against dictionary attacks". In: *Proceedings of the 19th International Conference on Theory and Application of Cryptographic Techniques*. EUROCRYPT'00. Bruges, Belgium: Springer-Verlag, 2000, pp. 139–155. ISBN: 3540675175.

[BM92] S. M. Bellovin and M. Merritt. "Encrypted Key Exchange: Password-Based Protocols Secure against Dictionary Attacks". In: *Proceedings of the IEEE Symposium on Security and Privacy*. Oakland, CA, USA: IEEE, May 1992, pp. 72–84. DOI: 10.1109/RISP.1992.213269.

[BCP03] Emmanuel Bresson, Olivier Chevassut, and David Pointcheval. "Security Proofs for an Efficient Password-Based Key Exchange". In: *Proceedings of the 10th ACM Conference on Computer and Communications Security*. Washington D.C., USA: ACM, Oct. 2003, pp. 241–250. DOI: 10.1145/948109.948142.

[Can+05] Ran Canetti et al. "Universally Composable Password-Based Key Exchange". In: *Advances in Cryptology – EUROCRYPT 2005*. Ed. by Ronald Cramer. Vol. 3494. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, May 2005, pp. 404–421.

[Erw+20] Andreas Erwig et al. *Fuzzy Asymmetric Password-Authenticated Key Exchange*. Cryptology ePrint Archive, Paper 2020/987. 2020. URL: https://eprint.iacr.org/2020/987.

[HL19] Björn Haase and Benoît Labrique. "AuCPace: Efficient verifier-based PAKE protocol tailored for the IIoT". In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2019.2 (Feb. 2019), pp. 1–48. DOI: 10.13154/tches.v2019.i2.1-48. URL: https://tches.iacr.org/index.php/TCHES/article/view/7384.

[Han+25] Shuya Hanai et al. *Universally Composable Relaxed Asymmetric Password-Authenticated Key Exchange*. Cryptology ePrint Archive, Paper 2025/571. 2025. DOI: 10.1007/978-3-031-71073-5_13. URL: https://eprint.iacr.org/2025/571.

[JRX24] Jake Januzelli, Lawrence Roy, and Jiayu Xu. *Under What Conditions Is Encrypted Key Exchange Actually Secure?* Cryptology ePrint Archive, Paper 2024/324. 2024. URL: https://eprint.iacr.org/2024/324.

[MRR20] Ian McQuoid, Mike Rosulek, and Lawrence Roy. "Minimal Symmetric PAKE and 1-out-of-N OT from Programmable-Once Public Functions". In: *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. CCS '20. Virtual Event, USA: Association for Computing Machinery, 2020, pp. 425–442. ISBN: 9781450370899. DOI: 10.1145/3372297.3417870. URL: https://doi.org/10.1145/3372297.3417870.

[Šal21] Petra Šala. "Attacks and security proofs of authenticated key-exchange protocols". Theses. Université Paris sciences et lettres ; Université du Luxembourg, Sept. 2021. URL: https://theses.hal.science/tel-03992889.

[SGJ23] Bruno Freitas Dos Santos, Yanqi Gu, and Stanislaw Jarecki. "Randomized Half-Ideal Cipher on Groups with Applications to UC (a)PAKE". In: *Advances in Cryptology – EUROCRYPT 2023, Part V*. Ed. by Carmit Hazay and Martijn Stam. Vol. 14008. Lecture Notes in Computer Science. Springer, Cham, Apr. 2023, pp. 128–156. DOI: 10.1007/978-3-031-30620-4_5.

[Sho20] Victor Shoup. *Security analysis of SPAKE2+*. Cryptology ePrint Archive, Paper 2020/313. 2020. URL: https://eprint.iacr.org/2020/313.

[Xu25] Jiayu Xu. *UC-Security of Encrypted Key Exchange: A Tutorial*. Cryptology ePrint Archive, Paper 2025/237. 2025. URL: https://eprint.iacr.org/2025/237.