SAPAKE: a smartphone assisted PAKE

Master Thesis

Master in *Sciences and Technologies*,
Specialty in *Mathematics*,
Track *Cryptology and Computer Security*.

Author
Giuseppe Guagliardo <giuseppe.guagliardo@etu.u-bordeaux.fr>

Supervisor
David Pointcheval <david.pointcheval@ens.fr>

Tutor
Gilles Zemor <gilles.zemor@u-bordeaux.fr>

October 22, 2015

Declaration of authorship of the document

I hereby certify that this material, which I now submit for assessment on the programme of study leading to the award of the Master in *Sciences and Technologies*, Specialty in *Mathematics* or *Computer Science*, Track *Cryptology and Computer Security*, is entirely my own work, that I have exercised reasonable care to ensure that the work is original, and does not to the best of my knowledge breach any law of copyright, and has not been taken from the work of others save and to the extent that such work has been cited and acknowledged within the text of my work.

Date and Signature

Abstract

This report presents a new two factor password based authenticated key exchange protocol that allows a client to create an exchange key with the help of his smart phone. The name of the protocol is "Smartphone assisted PAKE, simply SAPAKE.
Alongside the protocol, we show it is secure and we apply the games-sequence technique in the proof. Its safeness relies over the Diffie Hellman assumptions
Keywords: Password, encrypted key exchange, provable security, Diffie-Helmann, two factor authentication

TABLE OF CONTENTS

Introduction

In recent years the Cloud has come to play a pivotal role in the IT world: in a nutshell, cloud computing means that companies and private users need to connect to applications running on a set of shared servers instead of one dedicated server. The Cloud offers to its clients efficiency and reliability: a company will pay just for the space needed and will not need to manage a personal data server. Furthermore a company could quickly add capacity according to its needs. The Cloud computing offers a reduction on infrastructures' costs and maintenance. But with the public Clouds, privacy and secrecy are thorny issues. Clients store personal and sensible data in the Cloud as mails, bills and medical records and they should trust Cloud providers and communicate with them in a secure way. This is the reason why authentication protocols need to be designed. In order to communicate secretly over an unreliable public channel, two parties have to exchange a key without leaking any information about the secret. We will use and revisit the idea of password authenticated key exchange protocol: it basically consists in a cryptographic protocol that allows two entities who share the knowledge of a common secret (as a password) to mutually authenticate each other and establish a shared key, without revealing the secret in the process. In this thesis we are interested in creating a PAKE-protocol that allows a user to access his data on the Cloud with the knowledge of a low-entropy password and the help of a smartphone to generate several ephemeral shared keys with the server. We present a new two factor authentication protocol where the factors are a low entropy password and a personal secure device (as the smartphone). The protocol goes under the name of "Smartphone assisted-PAKE", abbreviated as SAPAKE. No client can access the data without his smart phone and vice-versa. This allows the client to access the cloud any time he desires without using always the same key, because every access has one fixed secret key and one ephemeral key. Moreover we want that the server authenticates itself to the client. Note that technically the smartphone is not always a secure device. We can consider instead a smartcard. Along with the protocol, we need to provide its security proof. In cryptography a security proof is generally a reduction from solving a well known problem to break the protocol. In this thesis we present the mathematical assumptions and the basic ideas of provable security for our proof based in the games-sequences technique. The present protocol and its proof are the fruit of a jointly work with professor Michel Abdalla and professor David Pointcheval, plus the Ph.D. student Mario Cornejo.

Part I

Key Exchange Protocols

1 Key Exchange Protocols

In network security key exchange protocols play an important role: they enable secure and private communication over an untrusted channel by setting up shared keys between two or more entities. Key exchange protocols have become necessary in the protection of transmitted data over an adversarially controlled network. In those protocols, the communication takes place between two entities (or pairs of entities) via messages, and upon completion, a shared key is generated. In the literature this key is known as *Session Key*. Furthermore the two parties, that wish to exchange a key, must activate a matching session. In the two party setting, the basic idea is to share a common key and use this latter to achieve privacy in their following communications. The most classical example is the Diffie-Helmann Key Exchange in the groups setting as shown in Fig 1 where $(\mathbb{G}, g, p)$ is a represented group with p a ℓ-bit prime number and g is the generator of the group. Both parties have to share the same initial information regarding the finite group.

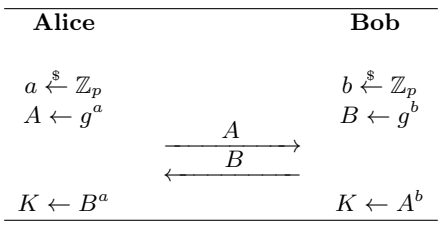

Fig. 1. A Diffie Hellman Key Exchange

The above protocol does not ensure authenticity. In order to add this latter to the protocol, the classical idea is to sign the message flows using a password. We will discuss the password authenticated key exchange protocols in the following sessions and upon those we will build our own protocol.

2 Notation

In the following we will use this notations. $(\mathbb{G}, g, p)$ is a represented group with p a ℓ-bit prime number and g the generator of the group. $x \xleftarrow{\$} \mathbb{Z}_p$ means that a random

value from $\mathbb{Z}_p$ is assigned to it. $\mathcal{H}$ is a hash function. In the random oracle model hash functions are modelled as perfectly random functions.

3 Authentication Protocols

Nowadays authentication has become one of the most important goal of modern cryptography. In order to add authenticity to a key exchange protocol, the different parties should know a common secret as a password before starting a session. PAKE protocols enable two entities, that want to communicate over an insecure channel, to generate an high entropy session key with just the knowledge of a low-entropy password easy to remember. This common password is used to hide the values sent in the message flows so that just the two parties sharing the password knowledge can create the same session key. The scenario foresees that one of the parties is a trusted server in which are stored the clients' passwords (generally ciphered). The aim of these protocols is also to make on-line guessing attack the only possible attack. In these attacks, the adversary must interact with the system to verify whether its guess is correct. But usually these systems rely on a policy of invalidating or blocking the use of a password after a certain number of failed attempts.

Authentication factors As to avoid mistakes and impersonations during access control, we can use various authentication means, possibly all together, that identify someone in a unique way: a secret information as a password, a biometric or user's belongings (fingerprints, a personal device) are the most well-known examples of such authentication factors for human beings. They represent the three types of human authentication factors admitted known under this name

- something you know: a low entropy password, a pin code.
- something you are: a user's biometric characteristic.
- something you have: a personal and secure device as a smartphone or a smartcard.

In order to authenticate in the web, today we have generally three different forms of authentication:

- Single Factor: just the knowledge of a password or a pincode.
- Two-Step (Or Multi-Step) : one single factor plus a code sent to the user (e.g. One time password for banking transitions).
- Two Factor (or Multi-Factor): different factors from the three types described above.

Typically the second step of Two Step authentication consists in receiving via e-mail or via sms a one time code to be used alongside the password. This one time code is classified under the " something you know" type. We are interested in designing a two factor protocol. Indeed the importance of multi-factor authentication is that an adversary must succeed in different kinds of theft to impersonate another person: he must acquire the knowledge of the password, a physical device and a fingerprint, for example. Thus, performing an attack will become even more struggling. Our protocol is a two factor PAKE protocol: in order to connect to the trusted server a client must use his secret and the session key is generated with another factor that the user possesses.

4 SAPAKE

Our authentication protocol builds upon previous password-based key exchange protocols in the three-party setting as described in [1–3]. Its description is given in Figure 2. U is a public information. The secret information are the low-entropy password pw, the high-entropy key stored on the mobile sk and $V = U^{sk+pw}$ stored in the server. Let us denote $s = sk + pw$. The hash function from $\{0,1\}^\star$ to $\{0,1\}^\ell$ are denoted $\mathcal{H}$: it is used to compute the session key and takes as input an honest transcript of the protocol. The protocol runs between three parties: the $\mathcal{C}$ who first communicates with the $\mathcal{SP}$ and then with $\mathcal{S}$ in order to compute the ephemeral key and the secret access key. The parties initially share the secret s which is not totally known neither by the client nor stored in the smartphone. The client starts the protocol sending a message to the smartphone. Then it picks at random a Diffie Hellmann exponent x_1 and responds with the value B: its part of the ephemeral key masked with the public value U to the secret sk. Then the client picks at random another exponent x_2 and computes the masked value $X^\star$ multiplying B for its part of the ephemeral key times the public value U to the password. In the end of the computing $X^\star$ will simply be $g^{x_1+x_2}V$. In this way, upon receipt of $X^\star$, the server instance can unmask the Diffie Hellman value X by dividing for its secret V. Then it picks at random an exponent y and sends to the client instance the masked value $Y^\star = YV$. In order to unmask $Y^\star$, the client communicates once again with its smartphone. Upon completion of the protocol, both the client and the server share the same Diffie-Helmann key g^{xy}. The protocol runs as showed in figure 2.

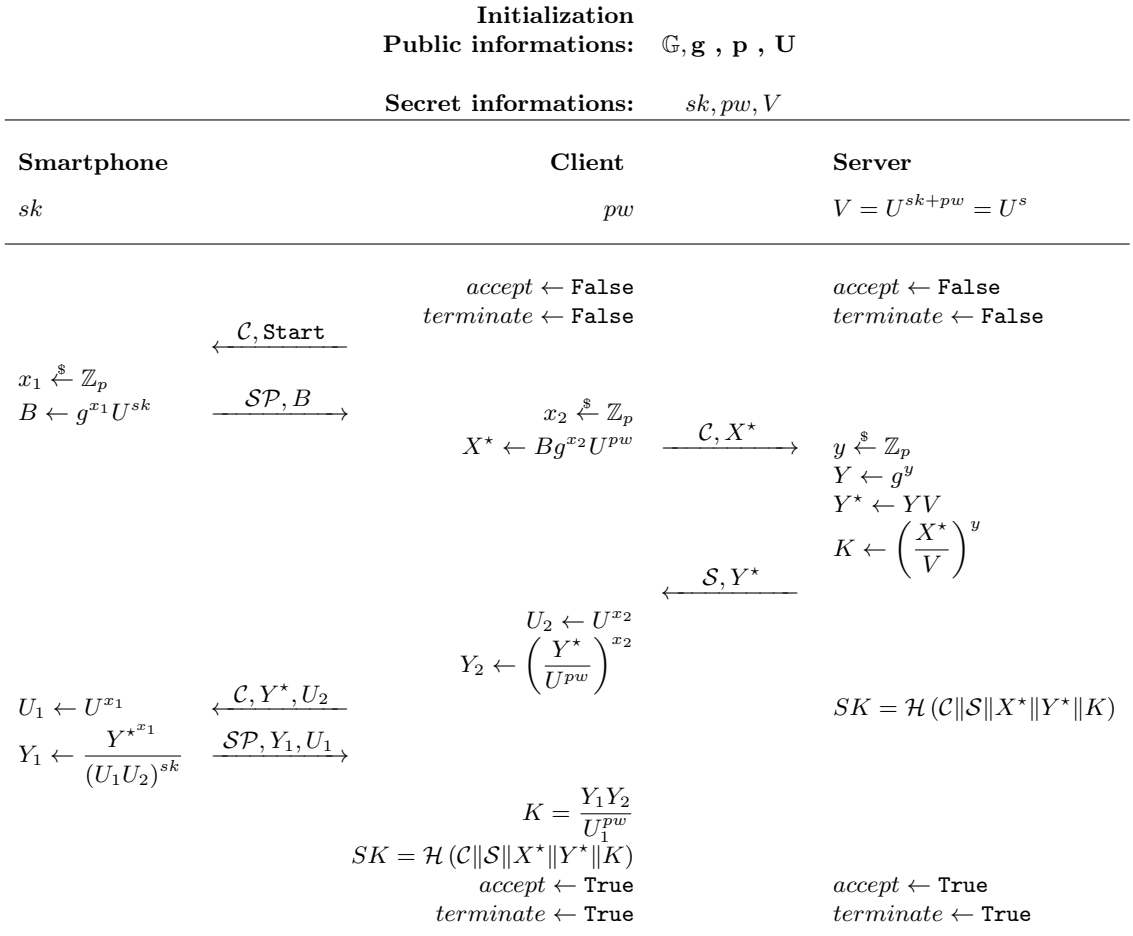

Fig. 2. An execution of the protocol run between a client and a server with the use of a smartphone

Note that at the end of the protocol the client and the server share the same key K : server computes

$$\begin{aligned} K &= \left(\frac{X^\star}{V}\right)^y = \left(\frac{Bg^{x_2}U^{pw}}{V}\right)^y \\ &= \left(\frac{g^{x_1+x_2}U^{sk+pw}}{V}\right)^y = \left(\frac{g^xV}{V}\right)^y = (g^x)^y = g^{xy} \end{aligned}$$

while the client computes

$$\begin{aligned} K &= \frac{Y_2Y_1}{U_1^{pw}} = \frac{\left(\frac{Y^\star}{U^{pw}}\right)^{x_2}\frac{Y^{\star x_1}}{(U_1U_2)^{sk}}}{U^{x_1pw}} \\ &= \frac{\left(\frac{g^{yx_2}U^{(sk+pw)x_2}}{U^{pwx_2}}\right)\frac{g^{yx_1}U^{(sk+pw)x_1}}{U^{(x_1+x_2)sk}}}{U^{x_1pw}} = \frac{g^{yx_2}U^{skx_2}\frac{g^{yx_1}U^{pwx_1}}{U^{x_2sk}}}{U^{x_1pw}} = \frac{g^{yx_2}g^{yx_1}U^{pwx_1}}{U^{x_1pw}} \\ &= g^{(x_1+x_2)y} = g^{xy} \end{aligned}$$

Part II

Provable Security

5 Provable Security

Any cryptosystem should be presented with the proof of its correctness and safeness. The purpose of provable security is to provide an evidence that the studied scheme is secure: this evidence has to be a mathematical guarantee. Provable security defines the security requirements of a cryptographic protocol: a mathematically rigorous theorem guarantees its security under certain conditions and given certain known mathematical assumptions. In order to certificate the security of the protocol we should define:

- The cryptographic scheme.
- The class of attackers and their goals.
- The model
- The condition where the attackers win
- The complexity assumptions
- Eventually a proof by reduction that no attacker can achieve the win condition in the defined model

The attackers' goals are multiples: the function one-wayness, the messages non-malleability or the key indistinguishability also known as semantic security. Since we are interested in key exchange protocols, here we will discuss about the semantic security: it consists in capturing the ability of the adversary to tell apart a real session key from a random one. Generally the proof consists in a reduction to a well known hard problem: if an attacker could break the scheme, then we could use the attacker to solve a problem known as hard. In order to provide the proof, we build a simulator that should solve the computational problem using the adversary as a sub-routine. Furthermore we must show that the view of the attacker in the simulation does not change from the view it has during a real attack. In the attack model we define what the adversary can do and what he can learn from an honest user of the scheme. Hereinafter we will describe the games sequences technique used in the proof and we will define the security model.

6 Games-sequence technique

In this part we introduce an useful technique to tame the complexity of security proofs. The proof is modelled as an attack game played between a *challenger* and an *adversary*. In order to prove the security using this technique,we should proceed as follows. First of all we define a particular event S where the adversary successes in winning the game and we define also a target event T in another game in which the adversary plays against a different challenger. Usually this target event consists in a well know hard assumption.

We define a sequences of games $Games_i$ for $i = 0, \ldots n$. All games operate in the same underlying probability space. $Game_0$ consists in the real attack against the protocol with respect to a given adversary. The construction of the games sequence defines the events S_i related respectively to $Games_i$. Let $S = S_0$. With the proof we want to show that the probability of S_i is negligibly close to those of S_{i-1} for $i = 0, \ldots n-1$ and that S_n is equal or negligibly close to the target probability. From this and the fact that n is constant, it follows that the probability of event S is negligibly close to that of T and so security is proved. Generally the target event is a well know hard assumption. In order to simplify the analysis of the changes between two successive games, successive games are very similar, typically with slightly different distribution probabilities. From experience there exists three different classes of transitions.

Identical simulations The two successive games change just conceptually and $\Pr[S_i] = \Pr[S_{i+1}]$. Typically the change consists in restating how certain quantities or variables can be computed in an equivalent way such that the adversary can not realize a change has been made. These transitions are useful to make the proof easier to follow.

Transitions depending on failure event The simulations of the two games are the same unless a specific failure event Ev occurs. If $\Pr[Ev]$ is negligible,we ignore it and apply the Shoup's lemma, [8].

Shoup's Lemma *Let A,B,E be events defined in some probability distribution, and suppose that* $A \wedge \neg E \iff B \wedge \neg E$. *Then* $|\Pr[A] - \Pr[B]| \leq \Pr[E]$
Proof. This comes from a simple computation:

$$\begin{aligned} |\Pr[A] - \Pr[B]| &= |\Pr[A \wedge \neg E] + \Pr[A \wedge E] - \Pr[B \wedge \neg E] - \Pr[B \wedge E]| \\ &= |\Pr[A \wedge E] - \Pr[B \wedge E]| \\ &\leq \Pr[E] \end{aligned}$$

The second equality follows from the hypothesis $A \wedge \neg E \iff B \wedge \neg E$ and the final inequality from the fact that $0 \leq \Pr[A \wedge \neg E], \Pr[B \wedge \neg E] \leq \Pr[E]$. □

So if we prove that event Ev occurs with a negligible probability, we prove also that the two successive games are negligibly close. Typically the difference between the two games consists in the way of computing some random variables.

Transitions based on indistinguishability In this kind of transition just a small change is made: if the adversary detects this change, then it is possible to construct an algorithm to distinguish between two distributions known indistinguishable. Let P_1 and P_2 be two computationally indistinguishable distributions. In order to prove that $|\Pr[S_i] - \Pr[S_{i+1}]|$ is negligible, we suppose that there exists a distinguish algorithm D that interpolates between the two games: when the input comes from distribution P_1, then D outputs 1 with $\Pr[S_i]$, otherwise with $\Pr[S_{i+1}]$. The assumption of the indistinguishability of the two distributions implies the thesis. Since the change made is really small, the construction of D is pretty obvious. Generally,we need to define the two games so that it is easy to merge them in a single "hybrid" game that takes an auxiliary input drawn from one of the indistinguishable distributions. In the end, the distinguisher simply runs this hybrid game and output 1 each time the appropriate event occurs.

Briefly, beginning from the real attack game, little changes need to be made between successive games with the aim of leaving the adversary unaware of these changes. The last game should lie over a well known hard assumptions in order to bond the probability to win the attack game.

7 Formal Model

In this section, we present the security model we will use for our security proof. We recall the security model for three party password-based authenticated key exchange protocol introduced in [2] built on [5,6] for key distribution schemes and [4] for password-authenticated key exchange.

7.1 Strong Authentication

A strong authentication is a protocol based on a challenge-response where one party *proves* its identity to another party (*the verifier*) by demonstrating knowledge of a secret known to be associated with that party, without revealing the secret itself to the verifier during the protocol [7]. In our case, we present a three-party protocol that achieves strong authentication, between a client, a *secure* device and an authentication server. The goal of the protocol is to establish an implicitly authenticated session key between the server and a client who has access to a secure device. Moreover, the client has a short password and also has access to a secure device that we model as a smartphone that stores a high entropy key. The server stores a value computed from the short password and the long key in such a way that the server does not know any of these values. The client, in order to authenticate himself, has to proof the knowledge of the secret and the possession of the smartphone before computing the session key, but the knowledge of the secret itself or the possession of the smartphone cannot lead to the computation of the session key. The security goal of the strong authentication is somehow related to three-party password-based key exchange. In our protocol the client and the server establish a session key shared between them with the help of the smartphone. If an attacker obtains access to the smartphone, its best chance is to guess the client's password by active (online) attempts. We also prove that the corruption of one of the parties and the learning of the secret stored cannot lead neither to the knowledge of the current session key nor to the previous ones.

7.2 Security Model

Protocol Participants. The participants in our protocol $\mathbf{P}$ are the client $C \in \mathcal{C}$, the server $S \in \mathcal{S}$ and the smartphone $SP \in \mathcal{SP}$. We denote by $\mathcal{U}$ the set of all participants such that $\mathcal{U} = \mathcal{C} \cup \mathcal{S} \cup \mathcal{SP}$. Each of them may have several instances called oracles involved in distinct, possibly concurrent executions of $\mathbf{P}$. Let $\mathcal{U}^i$ denote the instance i of particstudyipant $\mathcal{U}$, which can be either a client, an authentication server or a

smartphone. A client C holds a short entropy password pw_C drawn from the dictionary $\mathcal{D}$ of size N according to the uniform distribution, a smartphone SP holds a high entropy secret sk_{SP} and a server S holds the *derived secret* $sk_S[C, SP]$ for that client and smartphone. It is worth pointing out that, the key sk_{SP} and the password pw_C are interdependently drawn/generated but they are paired as soon as the server's key is derived. In other words, $pw_S[C, PS]$ can be derived from only one pair (sk_{SP}, pw_C) (except with negligible probability). An authentication server may handle different clients by storing a vector of derived passwords for multiple pairs key-password. From now on, we simple call sk_S to the derived secret key. All of these secret values are often called long-lived keys of the client, smartphone and server respectively.

Partnering. We based our definition of partnering as in [2] for three party setting, which is based on session identifications (*sid*). In order to all participants end up with the same session identification, the forwarding of messages may be required. More specifically, both a client and a server accept as long as the client communicates with *his* smartphone during the protocol.

Freshness. The goal of the adversary is to guess the bit b used during the `Test`-queries and as in [1] we opt not to embed the notion of freshness inside the definition of the oracles. However, in some cases, the adversary might trivially compute the session key and answer the `Test`-query whereas the adversary did not really break the semantic security. In addition we want to capture the three-party setting by modeling the lost/corruption of at most one long-lived key for each session.

We say that a instance is *fresh* if it has accepted and we allow the adversary only to perform tests on fresh instances. Moreover, in order to model the corruption of only one instance we limit the capability of the adversary making queries in the following cases:

- if C is corrupted, the adversary does not have access to query the SP instance for all the executions after the corruption was made, otherwise it could trivially break the semantic security. This models the knowledge of client's password but without the access to the smartphone;
- if SP is corrupted, C and S are still fresh. This models an attacker that has the smartphone and wants to authenticate himself by guessing the client password;
- if S is corrupted the attacker does not have access to query neither C nor SP instances for all the executions after the corruption was made. We model forward security by allowing the attacker to know the secret stored, and trying to break the semantic security of past sessions.

Communication Model. We do not assume any kind of authenticated or private channels between any of the participants. As usual the interaction between an adversary $\mathcal{A}$ and the protocol participants occurs only via oracle queries, which capture the capabilities of an adversary in a real attack. The types of oracles available to the adversary are:

- `Execute` $\left(\mathcal{C}^i, \mathcal{S}^j, \mathcal{SP}^k\right)$: This query consists of the messages exchanged during the honest execution of the protocol. This models passive attacks, where the adversary gets access by eavesdropping.
- `Send` $\left(\mathcal{U}^i, m\right)$: This query outputs the message that the instance $\mathcal{U}^i$ would generate upon receipt of message m. This models active attacks in which the adversary may intercept a message and then modify it, create a new one or simply forward it to the intended participant.

7.3 Security Notions

In order to define a notion of security for the protocol $\mathbf{P}$ we consider a game in which it is executed in presence of the adversary $\mathcal{A}$. In this game, we first draw a low-entropy short password pw from $\mathcal{D}$ and generate a high-entropy key sk, provide coin tosses and oracles to $\mathcal{A}$, and then run the adversary by letting it ask any number of queries as described above.

AKE security. In order to model the semantic security (privacy of the session key) we consider the $\mathbf{Game}^{\mathtt{ake}}(\mathcal{A}, \mathbf{P})$ in which an additional oracle to query is made available to the adversary, the `Test` $(\mathcal{U}^i)$ oracle. This oracle can be queried as many times as the adversary wants as soon as the instance $\mathcal{U}^i$ has accepted and holds the key exchange session. The `Test` query returns either the real session key if $b = 1$ and a random one if $b = 0$. In the later case, the same random key value should be returned when querying two partners.

- `Test` $\left(\mathcal{U}^i\right)$: this query tries to capture the ability of the adversary to tell apart a real session key from a random one. If no session key is defined for an instance $\mathcal{U}^i$, then return the symbol $\perp$. Otherwise, it flips a coin b (unique for the whole game) and return the real session key if $b = 1$ or a random key if $b = 0$.

The goal of the adversary is to guess the hidden bit b drawn in the `Test` query and we denote as Succ the event in which the adversary is successful and correctly guesses the value of b. The **AKE advantage** of an adversary $\mathcal{A}$ is then defined as:

$$\mathsf{Adv}_{\mathbf{P}}^{AKE}(\mathcal{A}) = 2\Pr[\,\mathsf{Succ}\,] - 1$$

The protocol **P** is said to be (t, ϵ)**-AKE-secure** if the advantage of the adversary is smaller than ϵ running with time t.

Authentication. The above property doest not guarantee the existence of a partner, but only the secrecy of the session key and that an adversary should not be able to learn information about the key. This is known as *implicit authentication*. In order to address this problem, one should add *authenticators* to achieve explicit authentication. Generally an hash function is used to compute those authenticators. Several protocols consider just *unilateral authentication* of the authentication server, by which the client can be ensured that it has in fact established a key with the authentication server instance it intended to. However in our protocol we opted for an implicit authentication of both instances

Forward-Secrecy. In our protocol we consider one more additional security property, *forward-secrecy*. A protocol is defined forward-secure if the security of a session key between two participants is preserved even if one of these participants' session keys is compromised in the future. We introduce a new type of query, the `Corrupt`, to model this additional property:

- `Corrupt` ($\mathcal{U}$): This query returns to the adversary the long-lived key hold by the participant $\mathcal{U}$: the password pw_C for the client, the key sk_{SP} for the smartphone and the secret value $pw_S[C, PS]$ for the server. We assume the weak corruption model, in which the internal states of the instances are not returned to the adversary. This can be seeing as to learn the long-lived key by coercing the participant, than completely compromising the machine by using a trojan or similar.

In order to model the forward-secrecy we state all the executions completed after the `Corrupt` query to be always answered with the real session key, independently of the bit b of the `Test` query. This notion is defined in order to avoid cases in which adversary can trivially break the security of the protocol. We define Succ to the event in which the adversary successfully guesses the hidden bit b used by `Test` oracle. We define the **FS-AKE advantage** as follows:

This notion can be tightened up a little, as follows: after the `Corrupt` query, the session keys exchanged by participants who behave honestly are still fresh, this is, when the adversary is not impersonating any of the participants, like for all `Execute` call.

$$\mathsf{Adv}_{\mathbf{P}}^{\mathbf{ake\text{-}fs}}(\mathcal{A}) = 2\Pr\left[\,\mathsf{Succ}\,\right] - 1$$

The protocol $\mathbf{P}$ is said to be (t, ϵ)-**FS-AKE-secure** if $\mathcal{A}$'s advantage is smaller than ϵ for any adversary $\mathcal{A}$ running with time t.

8 Computational Assumptions

In this section we recall some definitions of standard Diffie-Hellman assumptions and we introduce the building blocks for our protocol. We consider a finite multiplicative cyclic group $\mathbb{G} = \langle g \rangle$ of prime order q.

Computational Diffie-Hellman Assumption (CDH). The CDH assumption states that given g^x and g^y, where x and y were drawn at random from $\mathbb{Z}_q$, it is hard to compute g^{xy}. This can be defined considering the following experiment:

$$\begin{gathered}\mathsf{Exp}_{\mathbb{G}}^{\mathsf{cdh}}\left(\mathcal{A}, g^x, g^y\right)\\ g^z \leftarrow \mathcal{A}\left(g^x, g^y\right)\\ \textbf{if } g^z = g^{xy} \textbf{ then } b \leftarrow 1\\ \textbf{else } b \leftarrow 0\\ \textbf{return } b\end{gathered}$$

We denote by $\mathsf{Succ}^{\mathsf{cdh}}(\mathcal{A})$ the success probability of the adversary $\mathcal{A}$ in computing g^{xy}, and more generally, $\mathsf{Succ}^{\mathsf{cdh}}(t)$ is the best success probability an adversary can get within time t.

Chosen-Basis computation Diffie-Hellman (CCDH). This is a variation of the CDH problem. It considers an adversary who is given three random element of $\mathbb{G}$: U, V and X. The goal is to find a triple (Y, u, v) such that $u = \mathsf{CDH}(X, Y)$ and $v = \mathsf{CDH}\left(\frac{X}{U}, \frac{Y}{V}\right)$ at the same time. In order to solve this assumption, the adversary may be successfully compute either u (for example $Y = g$ and $u = X$) or v (for example $Y = gV$ and $v = \frac{X}{U}$), but not both.

$$\begin{gathered}\mathsf{Exp}^{\mathsf{ccdh}}_{\mathbb{G}}(\mathcal{A}, X, U, V)\\ (Y, u, v) \leftarrow \mathcal{A}(X, U, V)\\ u' \leftarrow \mathsf{CDH}(X, Y)\\ v' \leftarrow \mathsf{CDH}\left(\frac{X}{U}, \frac{Y}{V}\right)\\ \textbf{if } u = u' \textbf{ or } v = v' \textbf{then } b \leftarrow 1\\ \textbf{else } b \leftarrow 0\\ \textbf{return } b\end{gathered}$$

We denote by $\mathsf{Succ}^{\mathsf{ccdh}}(\mathcal{A})$ the success probability of the adversary $\mathcal{A}$ in guessing the bit b, and more generally, $\mathsf{Succ}^{\mathsf{ccdh}}(t)$ is the best success probability an adversary can get it within time t. In fact, solving this problem is equivalent to solving the underlying CDH problem. This equivalence is proved in [3].

Password-Based Chosen-Basis Computaional Diffie-Hellman (PCCDH). We recall the PCCDH problem introduced in [3]. It is a variation of CDH under password-based settings. Let $\mathcal{D}$ be a dictionary with n equally likely password values and let $\mathcal{P}$ be a injective map from $\mathcal{P}$ from $\{1, \dots n\}$ into $\mathbb{Z}_p$. The adversary runs two stages. In the first one, the adversary is given as input two random elements $U \in \mathbb{G}$ and $X \in \mathbb{G}$ and $\mathcal{P}$. The adversary outputs a value $Y \in \mathbb{G}$ (as we call, the chosen-basis). Next we chose a random password $k \in \{1, \dots, n\}$ and give it to the adversary. We compute the mapping $r = \mathcal{P}(k)$ of the password k. The goal of the adversary in the second stage is to output the value $K = \mathsf{CDH}_{g,\mathbb{G}}(X, Y/U^r)$.

$$\begin{gathered}\mathsf{Exp}^{\mathsf{pccdh}}_{\mathbb{G}}(\mathcal{A}, X, U, \mathcal{P})\\ (Y', st) \leftarrow \mathcal{A}(\mathsf{FIND}, X, U, \mathcal{P})\\ k \xleftarrow{\$} \{1, \dots n\}\ ;\ r \leftarrow \mathcal{P}(k)\\ K \leftarrow \mathcal{A}(\mathsf{GUESS}, st, k)\\ \textbf{if } K = \mathsf{CDH}(X, Y) \textbf{ then } b \leftarrow 1\\ \textbf{else } b \leftarrow 0\\ \textbf{return } b\end{gathered}$$

The idea behind the PCCDH assumption is that no adversary can do much better than guess the password with probability $1/n$, that is, $\mathsf{Succ}^{\mathrm{pccdh}}(t, n)$ cannot be significantly larger than $1/n$, where n is the size of the dictionary. This should be the case as long as the CDH assumption holds, as proved in [3].

Part III

Security Proof

9 Security Analysis

Our security analysis will be split into three different scenarios. We will consider three different attacks to our protocol:

- The adversary obtained the client's password. In this case the adversary will try to impersonate the client and authenticate himself to the server instance.
- The adversary stole the client's smartphone. With the help of the smartphone, $\mathcal{A}$ will try to break the secrecy of the communication between the client and the server.
- The adversary found out the secret V store in the server. In this case $\mathcal{A}$ will try to discover the session key of the previous honest communications between the client and the server using the value V.

This will lead us to our main result which is a secure protocol even when one of the parties has been compromised and impersonated. In the three theorems we simulate the protocol as follows, assuming that one corruption has made since the beginning. This means that the `Corrupt` query is not available anymore to the adversary.

$\mathcal{H}$ oracles	On a hash query $\mathcal{H}(q)$ (respectively $\mathcal{H}'(q)$), such that a record (i, q, r) appears in $\Lambda_{\mathcal{H}}$ (resp.$\Lambda_{\mathcal{H}'}$), the answer is r. Otherwise one chooses a random element $r \in \{0,1\}^l$, answers with it and add the record (q, r) to $\Lambda_{\mathcal{H}}$ (resp. $\Lambda_{\mathcal{H}'}$)

Fig. 3. Simulation of random oracle H

Other queries

An `Execute` $\left(\mathcal{SP}^k, \mathcal{C}^i, \mathcal{S}^j\right)$- query is processed as follows:

- $\alpha, \beta, y, B \xleftarrow{\$} \mathbb{Z}_p$
- $X \leftarrow g^{\alpha+\beta} \quad Y \leftarrow g^y \quad U_1 \leftarrow U^\alpha \quad U_2 \leftarrow U^\beta$
- $X^\star \leftarrow XV \quad Y^\star \leftarrow YV \quad Y_1 \leftarrow \dfrac{{Y^\star}^\beta}{(U_1 U_2)^{sk}}$
- $K \leftarrow X^y$
- $SK = \mathcal{H}\left(C \| S \| X^\star \| Y^\star \| K\right)$
- Then the query is answered with

$$\left(\mathcal{C}^i, \text{Start}\right), \left(\mathcal{SP}^k, B\right), \left(\mathcal{C}, X^\star\right), \left(\mathcal{S}, Y^\star\right), \left(C^i, Y^\star, U_2\right), \left(\mathcal{SP}^k, Y_1, U_1\right)$$

A `Test` $\left(U^i\right)$-query first gets SK for the instance $\mathcal{U}$, then flips a coin b. If $b = 1$, outputs SK, otherwise a random value SK of the same length.

Fig. 4. Simulation of `Execute` and `Test` queries

`Send`-query to $\mathcal{C}$

This is how we respond to the `Send`-queries to a client instance:

- `Send` $\left(\mathcal{C}^i, \text{Start}\right)$: if the instance $\mathcal{C}^i$ already exists, then abort; otherwise
  - Set terminate and accept to false and begin the instance
  - return $\left(\mathcal{C}^i, \text{Start}\right)$
- `Send` $\left(\mathcal{C}^i, \left(\mathcal{SP}^k, B\right)\right)$:
  - $\alpha \xleftarrow{\$} \mathbb{Z}_p \quad X^\star \leftarrow B g^\alpha U^{pw}$
  - return $\left(\mathcal{C}^i, X^\star\right)$
- `Send` $\left(\mathcal{C}^i, \left(\mathcal{S}^j, Y^\star\right)\right)$:
  - $U_2 \leftarrow U^\alpha$
  - return $\left(\mathcal{SP}^k, \left(\mathcal{C}^i, Y^\star, U_2\right)\right)$
- `Send` $\left(\mathcal{C}^i, \left(\mathcal{SP}^k, Y_1, U_1\right)\right)$:
  - $K' \leftarrow \dfrac{Y_1 {Y^\star}^{x_2}}{U_1^{pw} U_2^{pw}}$
  - Compute the session key via an $\mathcal{H}$ call.
  - $SK = \mathcal{H}\left(\mathcal{C} \| \mathcal{S} \| X^\star \| Y^\star \| K'\right)$
  - Then the instance accepts and terminates.

Fig. 5. Simulation of `Send` query

Send-query to $\mathcal{S}$	This is how we respond to the Send-queries to a server instance : - Send $\left(S^j, \left(C^i, X^\star\right)\right)$: • $Y^\star \leftarrow g^\theta V$ • $K \leftarrow \left(\frac{X^\star}{V}\right)^\theta$ • Compute the session key via an $\mathcal{H}$ call. • $SK = \mathcal{H}\left(C \| S \| X^\star \| Y^\star \| K\right)$ • Then the query is answered with $\left(S^j, Y^\star\right)$ and the instance accepts and terminates.

Fig. 6. Simulation of Send query

Send-query to $\mathcal{SP}$	This is how we respond to the Send-queries to a smartphone instance : - Send $\left(\mathcal{SP}^k, \left(C^i, \text{Start}\right)\right)$: • $\beta \xleftarrow{\$} \mathbb{Z}_p$, $B \leftarrow g^\beta U^{sk}$ • return $\left(\mathcal{SP}^k, B\right)$ - Send $\left(\mathcal{SP}^k, \left(C^i, Y^\star, U_2\right)\right)$: • $U_1 \leftarrow U^\beta$, $Y_1 \leftarrow \frac{{Y^\star}^\beta}{\left(U_1 U_2\right)^{sk}}$ • returns $\left(\mathcal{SP}^k, Y_1, U_1\right)$ and the instance terminates.

Fig. 7. Simulation of Send query

Theorem 1 *Let $G = (\mathbb{G}, g, q)$ be a represent group of prime order q and let $\mathcal{D}$ be a uniformly distributed dictionary of size $\#\mathcal{D}$. Let SAPAKE describe the two factor protocol associated with these primitives as defined in Figure 2 and in presence of an adversary that knows the password pw of the client. Then*

$$\begin{aligned}\mathsf{Adv}^{\mathsf{ake}}_{SAPAKE}(\mathcal{A}) \leq \frac{q_{\mathcal{H}}^2}{2^{\ell+1}} + \frac{(q_{exe} + q_{Send})^2}{p} \\ + (q_{\mathcal{H}})\, \boldsymbol{Adv}^{CDH}_{\mathbb{G}}(t) + (q_{\mathcal{H}} q_{Send})\, \boldsymbol{Adv}^{CDH}_{\mathbb{G}}(t) \\ + (q_{fake\text{-}server})\, \boldsymbol{Adv}^{CDH}_{\mathbb{G}}(t)\end{aligned}$$

Proof. In this proof, we define a sequences of games starting at the real game $\mathbf{G}_0$ and ending up at $\mathbf{G}_n$. For each game $\mathbf{G}_i$ we define the following event:

- $\mathbf{S_i}$ for semantic security, which occurs if the adversary correctly guesses the bit b involved in the *Test*-query;

In this case, we assume that the adversary knows the low-entropy password pw: the adversary can act as if he were the client. Since the adversary knows the password, he has not access to the communications with the smartphone: this means that the `Send`-queries to the smartphone are not allowed anymore. Furthermore it can ask `Test`-queries just to a $\mathcal{S}$ instance.

Game $\mathbf{G}_0$: This corresponds to the real attack which starts by choosing a random password pw and a smartphone key sk such that $V = U^{sk+pw}$. In this game the adversary $\mathcal{A}$ can make all the queries. By definition we have:

$$\mathsf{Adv}^{\mathsf{ake-fs}}_{\mathbf{G}_0}(\mathcal{A}) = 2 \times \Pr[\mathbf{S_0}] - 1$$

Game $\mathbf{G}_1$: In this game we simulate the oracle $\mathcal{H}$ as showed in figure 2 as usual by maintaining hash lists. In addition to these, we also simulate the private hash function $\mathcal{H}'$ to be used in later games. We finally simulate all the instances, as the real players would do, for the `Send`-queries , the `Execute`-queries and the `Test`-queries as shown in figure 3,4,5 and 6. From this simulation we can easily see that the game is perfectly indistinguishable from the previous one.

$$\mathsf{Adv}_{\mathbf{G}_1}(\mathcal{A}) = \mathsf{Adv}_{\mathbf{G}_0}(\mathcal{A})$$

Game $\mathbf{G}_2$: We first simulate all the oracles as in the previous game, except that we halt all executions in which some collision appear on the random oracle $\mathcal{H}_0$.
This probability is bounded by the birthday paradox:

$$\Pr\left[\,\mathsf{Coll}_H\,\right] \leq \frac{q_{\mathcal{H}}^2}{2^{\ell+1}}$$

Game $\mathbf{G}_3$: For a better and easier analysis in the following, in this game we simulate all the oracles as the previous game, except that we halt all executions in which some collisions appear on the transcripts. The absence of collisions will be useful in the following games.

- collisions on the partial transcripts

$$\left((\mathcal{C}, X^\star), (\mathcal{S}, Y^\star)\right)$$

  Note that transcripts in an active attacks via `Send`-queries involve at least one honest party because the adversary needs to interact with an honest party. Thus at least one of the message in the transcripts is truly uniformly distributed while in the `Execute`-queries all of them are uniformly distributed.

This probability is bounded by the birthday paradox:

$$\Pr\left[\,\mathsf{Coll}_3\,\right] \leq \frac{\left(q_{\mathtt{exe}} + q_{\mathtt{Send}}\right)^2}{p}$$

Game $\mathbf{G}_4$: We bound in this game the success probability of the adversary in passive attacks. To do so, we modify the way of computing the session key using the private oracles $\mathcal{H}'$ EXECUTE-queries. More precisely we compute the session key using the private oracle.

$$SK = \mathcal{H}'\left(\mathcal{C}\|\mathcal{S}\|X^\star\|Y^\star\right)$$

As results, SK becomes totally independent of the hash function and of the Diffie-Hellman keys K . $\mathbf{G_4}$ and $\mathbf{G_3}$ are indistinguishable unless the adversary queries $\mathcal{H}\left(\mathcal{C}\|\mathcal{S}\|X^\star\|Y^\star\|K\right)$ for some protocol execution transcript $(\mathcal{C}\|\mathcal{S}\|X^\star\|Y^\star)$. Let us denote this event as `AskHPassive`. In order to evaluate the probability of this event, we change the simulation in the game. Since we do not need to compute the two keys K for an `Execute`-query, we can introduce a CDH instance (Γ, Θ) in the simulation: for the K we simulate X as $g^{\alpha+\beta}\Gamma$ and Y as $g^y\Theta$. If event `AskHPassive` occurs, one can extract:

$$\begin{aligned}
K = \mathrm{CDH}\left(X, Y\right) &= \mathsf{CDH}\left(g^{\alpha+\beta}\Gamma, g^y\Theta\right) = \mathsf{CDH}\left(g^{\alpha+\beta}\Gamma, g^y\right)\mathsf{CDH}\left(g^{\alpha+\beta}\Gamma, \Theta\right) \\
&= \mathsf{CDH}\left(g^{\alpha+\beta}, g^y\right)\mathsf{CDH}\left(\Gamma, g^y\right)\mathsf{CDH}\left(g^{\alpha+\beta}, \Theta\right)\mathsf{CDH}\left(\Gamma, \Theta\right) \\
&= g^{(\alpha+\beta)y}\cdot\Gamma^y\cdot\Theta^{\alpha+\beta}\cdot\mathsf{CDH}\left(\Gamma, \Theta\right)
\end{aligned}$$

This means that K is in the list of the queries asked to $\mathcal{H}$. In order to realize that the game has been changed, $\mathcal{A}$ need to break the CDH assumptions. Thus a random guess leads to $\mathsf{CDH}(\Gamma, \Theta)$.

$$\Pr[\,\mathtt{AskHPassive}\,] \leq (q_{\mathcal{H}})\,\mathbf{Adv}^{\mathsf{CDH}}_{\mathbb{G}}(t)$$

Game $\mathbf{G}_5$: Let us consider in this game passive attacks via `Send`-queries, in which the adversary simply forwards the messages it receives from the oracle instances. As in the previous game we use the private oracle $\mathcal{H}'$ to compute the session key whenever the transcript $(\mathcal{C}\|\mathcal{S}\|X^\star\|Y^\star)$ has been generated by the oracle instances. We can safely do this because we have eliminated collisions in game $\mathbf{G_3}$. Like in $\mathbf{G_4}$, the results become totally independent of the hash function and the Diffie-Hellman keys. This game is indistinguishable from the previous one unless $\mathcal{A}$ asks the hash function $\mathcal{H}'$ on $(\mathcal{C}\|\mathcal{S}\|X^\star\|Y^\star\|K)$. We denote this bad event `AskHPassiveSend`. To evaluate the probability that the latter occurs, we slightly change the simulation without affecting the view of the adversary: we pick at random one of the `Send` $(\mathcal{C}^i, \mathtt{Start})$-queries being asked to $\mathcal{C}$, hoping is one of the sessions where the adversary simply forwards the queries. We introduce in this session a CDH instance (Θ, U), where U is the public value. To such query we simulate $X^\star = g^{\alpha+\beta}$ for random α and β. On the server side, we also change the simulation of $Y^\star$ whenever the latter receives an $X^\star = g^{\alpha+\beta}$: $Y^\star = \Theta$ with a random Θ and we compute the session key with the private oracle . Once again if the client receives from the server an $Y^\star = \Theta$, we simulate the U_2 as U^α using the same α as above. If the event `AskHPassiveSend` occurs, then one can extract from $\Lambda_{\mathcal{H}}$

$$\begin{aligned} K &= \mathsf{CDH}\left(\frac{X^\star}{V}, Y^\star\right) = \mathsf{CDH}\left(\frac{g^{\alpha+\beta}}{V}, \Theta\right) \\ &= \frac{\mathsf{CDH}\left(g^{\alpha+\beta}, \Theta\right)}{\mathsf{CDH}(V, \Theta)} = \frac{\Theta^{\alpha+\beta}}{\mathsf{CDH}(V, \Theta)} = \frac{\Theta^{\alpha+\beta}}{\mathsf{CDH}(U, \Theta)^{pw+sk}} \end{aligned}$$

Thus,

$$\Pr[\,\mathtt{AskHPassiveSend}\,] \leq (q_{\mathcal{H}} q_{\mathtt{Send}})\,\mathbf{Adv}^{\mathsf{CDH}}_{\mathbb{G}}(t)$$

Game $\mathbf{G}_6$: In this game we finally try to bound the advantage of the adversary in the active attacks via `Send`-queries when the adversary impersonates the client. In order to do so we change the simulation of the `Send`-queries to the client instance and replace $\mathcal{H}$ by the private one. Each time the server instance receives a $\overline{X^\star}$ not

simulated and generated by the adversary, we reply with $Y^\star = g^y\Theta$. Here we do not use the secret value V and we introduce a new variable Θ in order to have a CDH instance. We also compute the session key as $SK = \mathcal{H}'(\mathcal{C}\|\mathcal{S}\|\overline{X^\star}\|Y^\star)$ and set the state to accept and then terminate. Now in every case to compute the session key we use the random oracle $\mathcal{H}'$ that is private to the simulation. This means that the bit b involved in the `Test`-query cannot be guessed by $\mathcal{A}$, better than at random for each attempt. Thus $\Pr[S_6] = \frac{1}{2}$. This game is indistinguishable from the previous one unless $\mathcal{A}$ queries the hash functions $\mathcal{H}$ on an input $\left(\mathcal{C}\|\mathcal{S}\|\overline{X^\star}\|Y^\star\|K\right)$ for some execution transcript $\left(\mathcal{C}\|\mathcal{S}\|\overline{X^\star}\|Y^\star\right)$. We denote this bad event `AskHActive_wServer`:

$$|\Pr[A_6] - \Pr[A_5]| \leq \Pr[\texttt{AskHActive\_wServer}]$$

Note that even with the knowledge of the password, since sk is an high-entropy variable, it is impossible for the adversary to guess it and then retrieve the value V.
Even if $\mathcal{A}$ has created $\overline{X^\star}$, in order to query $\left(\mathcal{C}\|\mathcal{S}\|\overline{X^\star}\|Y^\star\|K\right)$ to the hash oracle, the adversary needs to compute $K = \mathsf{CDH}\left(Y^\star, \overline{X^\star}\right) = \mathsf{CDH}\left(g^y\Theta, \overline{X^\star}\right) = \overline{X^\star}^y\mathsf{CDH}\left(\Theta, \overline{X^\star}\right)$. If $\mathcal{A}$ find the correct K, then K is in the list of the $\mathcal{H}$ queries. Then from his answer we can solve the $\mathsf{CDH}\left(\Theta, \overline{X^\star}\right)$

$$\Pr[\texttt{AskHActive\_wServer}] \leq (q_{\textit{fake-client}})\,\mathbf{Adv}^{\mathsf{CDH}}_{\mathbb{G}}(t)$$

This concludes the proof of theorem 1.

Theorem 2 *Let $G = (\mathbb{G}, g, q)$ be a represent group of prime order q and let $\mathcal{D}$ be a uniformly distributed dictionary of size $\#\mathcal{D}$. Let SAPAKE describe the two factor protocol associated with these primitives as defined in Figure 2 and in presence of an adversary that knows the high-entropy key sk stored in the smartphone. Then*

$$\begin{aligned}\mathsf{Adv}^{\mathsf{ake}}_{SAPAKE}(\mathcal{A}) \leq \frac{q_{\mathcal{H}}^2}{2^{\ell+1}} + \frac{(q_{exe} + q_{Send})^2}{p} + (q_{\mathcal{H}})\, \boldsymbol{Adv}^{CDH}_{\mathbb{G}}(t) \\ + (q_{\mathcal{H}} q_{Send})\, \boldsymbol{Adv}^{CDH}_{\mathbb{G}}(t) \\ + (q_{fake\text{-}server} + q_{fake\text{-}client})\, \boldsymbol{Adv}^{PCCDH}_{\mathbb{G}}(t)\end{aligned}$$

Proof. As defined in the model, with the corruption of the smartphone, the $\mathcal{C}$ and $\mathcal{S}$ instances remain fresh and $\mathcal{A}$ can perform all the queries. The corruption of the smartphone corresponds to the scenario wherein $\mathcal{A}$ has stolen the device. In this proof, we define a sequences of games starting at the real game $\mathbf{G}_0$ and ending up at $\mathbf{G}_n$. For each game $\mathbf{G}_i$ we define the following event:

- $\mathbf{S_i}$ for semantic security, which occurs if the adversary correctly guesses the bit b involved in the *Test*-query;

Game $\mathbf{G}_0$: This corresponds to the real attack which starts by choosing a random password pw and a smartphone key sk such that $V = U^{sk+pw}$. Remark that $\mathcal{A}$ knows the secret sk. In this game the adversary $\mathcal{A}$ can make all the queries. By definition we have :

$$\mathsf{Adv}^{\mathsf{ake-fs}}_{\mathbf{G}_0}(\mathcal{A}) = 2 \times \Pr[\mathbf{S_0}] - 1$$

Game $\mathbf{G}_1$: In this game we simulate the oracle $\mathcal{H}$ as showed in figure 2 as usual by maintaining hash lists. In addition to these, we also simulate the private hash function $\mathcal{H}'$ to be used in later games. We finally simulate all the instances, as the real players would do, for the `Send`-queries , the `Execute`-queries and the `Test`-queries as shown in figure 3,4,5 and 6. From this simulation we can easily see that the game is perfectly indistinguishable from the previous one.

$$\mathsf{Adv}_{\mathbf{G}_1}(\mathcal{A}) = \mathsf{Adv}_{\mathbf{G}_0}(\mathcal{A})$$

Game $\mathbf{G}_2$: We first simulate all the oracles as in the previous game, except that we halt all executions in which some collision appear on the random oracle $\mathcal{H}$.

This probability is bounded by the birthday paradox:

$$\Pr\left[\,\mathsf{Coll}_H\,\right] \leq \frac{q_{\mathcal{H}}^2}{2^{\ell+1}}$$

Game $\mathbf{G}_3$: For a better and easier analysis in the following, in this game we simulate all the oracles as the previous game, except that we halt all executions in which some collisions appear on the transcripts. The absence of collisions will be useful in the following games.

- collisions on the partial transcripts

$$\left((\mathcal{C}, X^\star), (\mathcal{S}, Y^\star)\right)$$

Note that transcripts in an active attacks via `Send`-queries involve at least one honest party because the adversary needs to interact with an honest party. Thus at least one of the message in the transcripts is truly uniformly distributed while in the `Execute`-queries all of them are uniformly distributed.

This probability is bounded by the birthday paradox:

$$\Pr\left[\,\mathsf{Coll}_3\,\right] \leq \frac{\left(q_{\mathtt{exe}} + q_{\mathtt{Send}}\right)^2}{p}$$

Game $\mathbf{G}_4$: We bound in this game the success probability of the adversary in passive attacks To do so , we modify the way of computing the session key using the private oracles $\mathcal{H}'$ EXECUTE-queries. More precisely we compute the session key using the private oracle.

$$SK = \mathcal{H}'\left(\mathcal{C}\|\mathcal{S}\|X^\star\|Y^\star\right)$$

As results, SK becomes totally independent of the hash function and of the Diffie-Hellman keys K. $\mathbf{G_4}$ and $\mathbf{G_3}$ are indistinguishable unless the adversary queries $\mathcal{H}\left(\mathcal{C}\|\mathcal{S}\|X^\star\|Y^\star\|K\right)$ for some protocol execution transcript $(\mathcal{C}\|\mathcal{S}\|X^\star\|Y^\star)$. Let us denote this event as `AskHPassive`. To evaluate the probability of this event, we change the simulation in the game. Since we do not need to compute the two keys K for an `Execute`-query, we can introduce a CDH instance (Γ, Θ) in the simulation: for the K we simulate X as $g^{\alpha+\beta}\Gamma$ and Y as $g^y\Theta$. If event `AskHPassive` occurs, one can extract:

$$\begin{aligned} K = \mathrm{CDH}\left(X, Y\right) &= \mathsf{CDH}\left(g^{\alpha+\beta}\Gamma, g^y\Theta\right) = \mathsf{CDH}\left(g^{\alpha+\beta}\Gamma, g^y\right)\mathsf{CDH}\left(g^{\alpha+\beta}\Gamma, \Theta\right) \\ &= \mathsf{CDH}\left(g^{\alpha+\beta}, g^y\right)\mathsf{CDH}\left(\Gamma, g^y\right)\mathsf{CDH}\left(g^{\alpha+\beta}, \Theta\right)\mathsf{CDH}\left(\Gamma, \Theta\right) \\ &= g^{(\alpha+\beta)y}\cdot\Gamma^y\cdot\Theta^{\alpha+\beta}\cdot\mathsf{CDH}\left(\Gamma, \Theta\right) \end{aligned}$$

This means that K is in the list of the queries asked to $\mathcal{H}$. In order to realize that the game has been changed, $\mathcal{A}$ need to break the CDH assumptions. Thus a random guess leads to $\mathsf{CDH}(\Gamma, \Theta)$.

$$\Pr[\,\mathtt{AskHPassive}\,] \leq (q_{\mathcal{H}})\,\mathbf{Adv}^{\mathsf{CDH}}_{\mathbb{G}}(t)$$

Game $\mathbf{G}_5$: Let us consider in this game passive attacks via `Send`-queries, in which the adversary simply forwards the messages it receives from the oracle instances. As in the previous game we use the private oracle $\mathcal{H}'$ to compute the session key whenever the transcript $(\mathcal{C}\|\mathcal{S}\|X^\star\|Y^\star)$ has been generated by the oracle instances. We can safely do this because we have eliminated collisions in game $\mathbf{G_3}$. Like in $\mathbf{G_4}$, the results become totally independent of the hash function and the Diffie-Hellman keys. This game is indistinguishable from the previous one unless $\mathcal{A}$ asks the hash function $\mathcal{H}'$ on $(\mathcal{C}\|\mathcal{S}\|X^\star\|Y^\star\|K)$. We denote this bad event `AskHPassiveSend`. In order to evaluate the probability that the latter occurs, we slightly change the simulation without affecting the view of the adversary: we pick at random one of the `Send` $\left(\mathcal{C}^i, \mathtt{Start}\right)$-queries being asked to $\mathcal{C}$, hoping is one of the sessions where the adversary simply forwards the queries. We introduce in this session a CDH instance (Θ, U), where U is the public value. To such query we simulate $X^\star = g^{\alpha+\beta}$ for random α and β. On the server side, we also change the simulation of $Y^\star$ whenever the latter receives an $X^\star = g^{\alpha+\beta}$: $Y^\star = \Theta$ with a random Θ and we compute the session key with the private oracle. Once again if the client receives from the server an $Y^\star = \Theta$, we simulate the U_2 as U^α using the same α as above. If the event `AskHPassiveSend` occurs, then one can extract from $\Lambda_{\mathcal{H}}$

$$\begin{aligned} K &= \mathsf{CDH}\left(\frac{X^\star}{V}, Y^\star\right) = \mathsf{CDH}\left(\frac{g^{\alpha+\beta}}{V}, \Theta\right) \\ &= \frac{\mathsf{CDH}\left(g^{\alpha+\beta}, \Theta\right)}{\mathsf{CDH}(V, \Theta)} = \frac{\Theta^{\alpha+\beta}}{\mathsf{CDH}(V, \Theta)} = \frac{\Theta^{\alpha+\beta}}{\mathsf{CDH}(U, \Theta)^{pw+sk}} \end{aligned}$$

Thus,

$$\Pr[\,\mathtt{AskHPassiveSend}\,] \leq (q_{\mathcal{H}} q_{\mathtt{Send}})\,\mathbf{Adv}^{\mathsf{CDH}}_{\mathbb{G}}(t)$$

Game $\mathbf{G}_6$: In this game we try to bound the advantage of the adversary in the active attacks via `Send`-queries against the client in which the adversary may have generated the inputs of the Hash oracle. In order to do so we change the simulation of the `Send`-queries to the client instance and replace $\mathcal{H}$ by the private one as showed in the figure 8.

Modified **Send**-queries

- On a query type $\texttt{Send}\left(\mathcal{C}^i, \left(\mathcal{SP}^k, B\right)\right)$ we reply with $\left(\mathcal{C}^i, X^\star = g^x\right)$ for a random $x \in \mathbb{Z}_p$. Here we do not use anymore the password pw and the distribution of $X^\star$ remains identical.
- On a query type $\texttt{Send}\left(\mathcal{C}^i, \left(\mathcal{S}^j, \overline{Y^\star}\right)\right)$, we compute the value U_2 as U^x with the same x for the above query. Then we respond with $\left(\mathcal{SP}^k, \left(\mathcal{C}^i, \overline{Y^\star}, U_2\right)\right)$ Since that for each protocol execution the value x_2 must to be random, the value U_2 will be always randomly chosen. Note that $\overline{Y^\star}$ has been generated by the adversary.
- On a query type $\texttt{Send}\left(\mathcal{C}^i, \left(\mathcal{SP}^k, Y_1, U_1\right)\right)$ we do not need to compute the keys K with the values received from the smartphone, thus we can simply compute the session key with the private hash function $\mathcal{H}'\left(\mathcal{C}\|\mathcal{S}\|X^\star\|\overline{Y^\star}\right)$. Then the instance accepts and terminates.

Fig. 8. Send simulation modification.

From the point of view of the adversary, this game is indistinguishable from the previous one unless $\mathcal{A}$ queries the hash functions $\mathcal{H}_i$ on an input $\left(\mathcal{C}\|\mathcal{S}\|X^\star\|\overline{Y^\star}\|K\right)$ for some execution transcript $\left(\mathcal{C}\|\mathcal{S}\|X^\star\|\overline{Y^\star}\right)$, where d $K = \mathsf{CDH}\left(\frac{X^\star}{V}, \overline{Y^\star}\right)$ where $\overline{Y}$ was generated by the adversary. We denote this bad event as `AskHActive_wClient`

$$|\Pr\left[S_6\right] - \Pr\left[S_6\right]| \leq \Pr\left[\texttt{AskHActive\_wClient}\right]$$

We will compute this probability after having computed the probability the adversary has in faking the client.
Note that we do not consider active attacks against the smartphone since we have assumed sk is already known by $\mathcal{A}$.
To compute the probability of the adversary in faking the server, we consider the event `AskHActive_wClient`. At this point we do not use the password anymore for the simulation of $X^\star$. If $\mathcal{A}$ queries to the hash oracle $\left(\mathcal{C}\|\mathcal{S}\|X^\star\|\overline{Y^\star}\|K\right)$ where $\overline{Y^\star}$ was created by $\mathcal{A}$, it must have computed K. In this case if $\mathcal{A}$ succeeds in this computation, we can use it to solve the PCCDH problem. Indeed we have given to $\mathcal{A}$ the values $X^\star$ and U and it has produced Y. Since we have not used the password, we can choose now one value for pw' and extract from the answer $K = \mathsf{CDH}\left(\frac{X^\star U^{-sk}}{U^{pw'}}, \overline{Y^\star}\right)$ solving the PCCDH problem. Thus,

$$\texttt{AskHActive\_wClient} \leq (q_{fake\text{-}server})\, \mathbf{Adv}_{\mathbb{G}}^{\mathsf{PCCDH}}(t)$$

Game $\mathbf{G}_7$: In this game we focus on the case where $\mathcal{A}$ impersonates the client against the server: $\overline{X^\star}$ was generated by the adversary while $Y^\star$ was simulated.

We change slightly the simulation of the `Send`-queries: each time the server instance receives a $\overline{X^\star}$ not simulated, we reply with an $Y^\star = g^y$ for a random y. Note that we do not use anymore the value V to mask $Y^\star$ and we do not use either the pw. We compute the session key with the hash $\mathcal{H}'$ and set the state to accept and then terminate.
At this point, we remark that in every protocol execution (with passive and/or active attacks) the session key is always computed with the random oracle $\mathcal{H}'$ that is private to the simulation. This means that the bit b involved in the `Test`-query cannot be guessed by $\mathcal{A}$, better than at random for each attempt. Thus $\Pr[S_7] = \frac{1}{2}$. This game is indistinguishable from the previous one unless $\mathcal{A}$ queries the hash functions $\mathcal{H}$ on an input $\left(\mathcal{C}\|\mathcal{S}\|\overline{X^\star}\|Y^\star\|K\right)$ for some execution transcript $\left(\mathcal{C}\|\mathcal{S}\|\overline{X^\star}\|Y^\star\right)$. We denote this bad event `AskHActive_wServer`:

$$|\Pr[S_7] - \Pr[S_6]| \leq \Pr[\texttt{AskHActive\_wServer}]$$

Even if $\mathcal{A}$ has created $\overline{X^\star}$, no information is leaked about the secret value V nor about the pw. As before we can just choose now a random password pw'. Thus, in order to query $\left(\mathcal{C}\|\mathcal{S}\|\overline{X^\star}\|Y^\star\|K\right)$ to the hash oracle, the adversary needs to compute $K = \mathsf{CDH}\left(\frac{Y^\star U^{-sk}}{U^{pw'}}, \overline{X^\star}\right)$ and from his answer we can solve the PCCDH problem.

$$\Pr[\texttt{AskHActive\_wServer}] \leq (q_{\textit{fake-client}})\, \mathbf{Adv}^{\mathsf{PCCDH}}_{\mathbb{G}}(t)$$

This concludes the proof of theorem 2.

Theorem 3 *Let $G = (\mathbb{G}, g, q)$ be a represent group of prime order q and let $\mathcal{D}$ be a uniformly distributed dictionary of size $\#\mathcal{D}$. Let SAPAKE describe the two factor protocol associated with these primitives as defined in Figure 2 and in presence of an adversary that knows the high-entropy key V stored in the server. Then*

$$\mathsf{Adv}^{\mathsf{ake-fs}}_{SAPAKE}(\mathcal{A}) \leq \frac{q^2_{\mathcal{H}_0}}{2^{\ell_0+1}} + \frac{\left(q_{\texttt{exe}} + q_{\texttt{Send}}\right)^2}{p} + (q_{\mathcal{H}})\, \boldsymbol{Adv}^{\mathsf{CDH}}_{\mathbb{G}}(t)$$

Proof. In this proof, we define a sequences of games starting at the real game $\mathbf{G}_0$ and ending up at $\mathbf{G}_n$. The goal of proof is to show that the adversary can not retrieve the session keys after he has obtained the knowledge of the secret value V. As explained in the model, he has no access to the client or the smartphone instances. So we just need to prove that even with the knowledge of V, it is impossible to guess the bit involved in an honest execution of the protocol. For each game $\mathbf{G}_i$ we define the following event:

- $\mathbf{S_i}$ for semantic security, which occurs if the adversary correctly guesses the bit b involved in the *Test*-query;

Game $\mathbf{G}_0$: This corresponds to the real attack which starts by choosing a random password pw and a smartphone key sk such that $V = U^{sk+pw}$. In this game the adversary $\mathcal{A}$ can make all the queries. By definition we have :

$$\mathsf{Adv}^{\mathsf{ake-fs}}_{\mathbf{G}_0}(\mathcal{A}) = 2 \times \Pr\left[\,\mathbf{S_0}\,\right] - 1$$

Game $\mathbf{G}_1$: In this game we simulate the oracle $\mathcal{H}$ as showed in figure 2 as usual by maintaining hash lists. In addition to these, we also simulate the private hash function $\mathcal{H}'$ to be used in later games. We finally simulate all the instances, as the real players would do, for the `Send`-queries , the `Execute`-queries and the `Test`-queries as shown in figure 3,4,5 and 6. From this simulation we can easily see that the game is perfectly indistinguishable from the previous one.

$$\mathsf{Adv}_{\mathbf{G}_1}(\mathcal{A}) = \mathsf{Adv}_{\mathbf{G}_0}(\mathcal{A})$$

Game $\mathbf{G}_2$: We first simulate all the oracles as in the previous game, except that we halt all executions in which some collision appear on the random oracle $\mathcal{H}_0$.
This probability is bounded by the birthday paradox:

$$\Pr\left[\,\mathsf{Coll}_H\,\right] \leq \frac{q^2_{\mathcal{H}_0}}{2^{\ell_0+1}}$$

Game $\mathbf{G}_3$: For a better and easier analysis in the following, in this game we simulate all the oracles as the previous game, except that we halt all executions in which some collisions appear on the transcripts. The absence of collisions will be useful in the following games.

- collisions on the partial transcripts

$$((\mathcal{C}, X^\star), (\mathcal{S}, Y^\star))$$

Note that transcripts in an active attacks via `Send`-queries involve at least one honest party because the adversary needs to interact with an honest party. Thus at least one of the message in the transcripts is truly uniformly distributed while in the `Execute`-queries all of them are uniformly distributed.

This probability is bounded by the birthday paradox:

$$\Pr[\,\mathsf{Coll}_3\,] \leq \frac{(q_{\mathtt{exe}} + q_{\mathtt{Send}})^2}{p}$$

Game $\mathbf{G}_4$: We bound in this game the success probability of the adversary in passive attacks. To do so , we modify the way of computing the session key using the private oracles $\mathcal{H}'$ EXECUTE-queries. More precisely we compute the session key using the private oracle.

$$SK = \mathcal{H}'(\mathcal{C}\|\mathcal{S}\|X^\star\|Y^\star)$$

As results, SK becomes totally independent of the hash function and of the Diffie-Hellman key K . $\mathbf{G_4}$ and $\mathbf{G_3}$ are indistinguishable unless the adversary queries $\mathcal{H}$ $(\mathcal{C}\|\mathcal{S}\|X^\star\|Y^\star\|K)$ for some protocol execution transcript $(\mathcal{C}\|\mathcal{S}\|X^\star\|Y^\star)$. Let us denote this event as `AskHPassive`.

In order to evaluate the probability of this event, we change the simulation in the game. Note that thanks to the knowledge of V the adversary can obtain respectively from $X^\star$ and $Y^\star$ the unmasked values X and Y. Since we do not need to compute the two keys K for an `Execute`-query, we can introduce a CDH instance (Γ, Θ) in the simulation: for the K we simulate X as $g^{\alpha+\beta}\Gamma$ and Y as $g^y\Theta$.

If event `AskHPassive` occurs, one can extract:

$$\begin{aligned} K = \mathrm{CDH}(X, Y) &= \mathsf{CDH}\left(g^{\alpha+\beta}\Gamma, g^y\Theta\right) = \mathsf{CDH}\left(g^{\alpha+\beta}\Gamma, g^y\right)\mathsf{CDH}\left(g^{\alpha+\beta}\Gamma, \Theta\right) \\ &= \mathsf{CDH}\left(g^{\alpha+\beta}, g^y\right)\mathsf{CDH}(\Gamma, g^y)\,\mathsf{CDH}\left(g^{\alpha+\beta}, \Theta\right)\mathsf{CDH}(\Gamma, \Theta) \\ &= g^{(\alpha+\beta)y} \cdot \Gamma^y \cdot \Theta^{\alpha+\beta} \cdot \mathsf{CDH}(\Gamma, \Theta) \end{aligned}$$

This means that K is in the list of the queries asked to $\mathcal{H}$. In order to realize that the game has been changed ,$\mathcal{A}$ need to break the CDH assumptions. The knowledge of the value V does not help to guess the bit b. Thus a random guess leads to $\mathsf{CDH}\,(\Gamma, \Theta)$.

$$\Pr\left[\,\texttt{AskHPassive}\,\right] \leq (q_{\mathcal{H}})\,\mathbf{Adv}_{\mathbb{G}}^{\mathsf{CDH}}\,(t)$$

Since in this scenario the adversary can just ask for honest protocol execution via `Execute`-queries , $\Pr\left[\,\mathbf{S_4}\,\right] = \dfrac{1}{2}$. Indeed in all the honest executions the session key is computed with the oracle $\mathcal{H}'$ private to $\mathcal{A}$ that can just guess at random. This is the proof for the forward secrecy: with the corruption of the server, the adversary has not access to other queries and the game stops here.
This concludes the proof of theorem 3.

Conclusion

Conclusion

In this report I have presented the work done during my internship at ENS in Paris under the supervision of David Pointcheval and Michel Abdalla. We have designed a new protocol and proved his safeness. The protocol consists in a two factors authentication password-based key exchange protocol and it is meant to be used in the communication with a cloud server. The authentication means are an easily human-memorable password and a personal device. This protocol enables to connect to the cloud and to create a shared cryptographic key to encrypt the data that will be sent in the cloud. Our purpose is to prove that the system is safe even if the device (modelled as a smartphone) is stolen or the password is known by a possible adversary. During the internship I had the chance to acquire and deepen some notions in the field of advanced cryptography. We redefined the protocol twice: indeed the first version of the protocol had some flaws and which we were able to detect thanks to the proof. Thus we made the changes needed to secure the protocol. We did not move to the phase of implementation of the protocol, still we think it is easily feasible. With the spread of the Cloud use, designing secure cryptographic protocols has become one of the primary tasks of a cryptographer. Then our goal is to prove that these protocols are safe. The scientific literature that I have studied during this experience provided me with a very useful technique to tame the complexity in a security proof.

References

1. Abdalla, M., Chevassut, O., Fouque, P.A., Pointcheval, D.: A simple threshold authenticated key exchange from short secrets. In: Roy, B.K. (ed.) ASIACRYPT 2005. LNCS, vol. 3788, pp. 566–584. Springer (Dec 2005) (Pages 5 and 13.)
2. Abdalla, M., Fouque, P.A., Pointcheval, D.: Password-based authenticated key exchange in the three-party setting. In: Vaudenay, S. (ed.) PKC 2005. LNCS, vol. 3386, pp. 65–84. Springer (Jan 2005) (Pages 5, 12, and 13.)
3. Abdalla, M., Pointcheval, D.: Simple password-based encrypted key exchange protocols. In: Menezes, A. (ed.) CT-RSA 2005. LNCS, vol. 3376, pp. 191–208. Springer (Feb 2005) (Pages 5 and 17.)
4. Bellare, M., Pointcheval, D., Rogaway, P.: Authenticated key exchange secure against dictionary attacks. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 139–155. Springer (May 2000) (Page 12.)
5. Bellare, M., Rogaway, P.: Entity authentication and key distribution. In: Stinson, D.R. (ed.) CRYPTO'93. LNCS, vol. 773, pp. 232–249. Springer (Aug 1994) (Page 12.)
6. Bellare, M., Rogaway, P.: Provably secure session key distribution: The three party case. In: 27th ACM STOC. pp. 57–66. ACM Press (May / Jun 1995) (Page 12.)
7. Menezes, A.J., Vanstone, S.A., Oorschot, P.C.V.: Handbook of Applied Cryptography. CRC Press, Inc., Boca Raton, FL, USA, 1st edn. (1996) (Page 12.)
8. Shoup, V.: Sequences of games: a tool for taming complexity in security proofs. Cryptology ePrint Archive, Report 2004/332 (2004), http://eprint.iacr.org/2004/332 (Page 10.)