# Authenticated Key Exchange protocol implemented in the Bounded-Retrieval model

**Abstract.** We introduce a two-party Authenticated Key Exchange (AKE) protocol and we discuss our implementation of it, in C programming language using the cryptographic library of the OpenSSL project. Our AKE protocol allows a client and a server, who share a huge secret file, to securely compute a shared key, providing client-to-server authentication, using only a small portion of the shared file. The protocol is secure against an attacker who retrieves a large portion of the shared file, prior of the execution of the protocol, and who actively controls the channel between the client and the server. Following the work of Cash et al. [25], our construction is based on a Weak Key Exchange (WKE) protocol, developed in the Bounded-Retrieval Model, and a Password-based Authenticated Key Exchange protocol showed secure in the Universally Composable framework. We focus our work both on the theoretical aspects of the problem, formally adapting the security analysis of the WKE protocol of [25] to our WKE protocol, and on the practical ones, analyzing concrete parameters and experimental evaluation of our implementation, to contribute to bridge the gap between theory and practice in the emerging field of Leakage-Resilient Cryptography.

**Key words:** Leakage-Resilient Cryptography, Authenticated Key Exchange, Bounded-Retrieval Model

## 1 Introduction

In 1976, the seminal work of Whitfield Diffie and Martin Hellman "New directions in cryptography" [28] introduced the Diffie-Hellman (DH) Key Exchange protocol, which allows two parties, with no prior knowledge of each other, to securely derive a shared secret key over an insecure channel, relying on the Computational Diffie-Hellman (CDH) assumption. In the classical DH Key Exchange, the public DH values of the parties are both sent in clear over the channel (i.e. without any encryption) and there is no authentication. For this reason, the DH Key Exchange suffers of the man-in-the-middle (MITM) attack, in which an adversary, who controls the channel between the parties, can force each party to share a valid secret key with her, while the parties believe to share this key among them. To perform such attack, an adversary simply needs to intercept the public DH values sent from a party to the other and to send, instead of it, her valid public DH values to the parties. Although the MITM attack looks simple, it is very serious, as, after such attack, the adversary can completely control all the communication between the parties. A countermeasures to this weakness of the DH protocol is *authentication*, which means that the parties running the protocol have to authenticate each other (or only in one direction) to check if they are sharing the secret key computed by the protocol with the other party or with some malicious adversary, accepting the key in the former case while aborting it in the latter one.

A simple way to realize an Authenticated Key Exchange (AKE) protocol consists in providing the parties with a password, a weak key, eventually not uniformly distributed, human-memorizable and with low entropy. Using this password, the parties can run a Password-based Authenticated Key Exchange (PAKE) protocol to perform the AKE protocol. In such setting, an adversary could try to use the low entropy of the password to break the security of the protocol, performing a so called dictionary attack, in which the adversary try to determine the password by checking all the possible values. Indeed, as the dictionary, the set of all possible (or most possible) values for the password, has a small size, the adversary can try all these values as password, with non-negligible probability of finding the right one. A dictionary attack can be performed *on-line* or *off-line*. On-line means that the adversary repeatedly runs the protocol and interacts with the other parties to verify if her guess of the password is correct. Hence an on-line attack is not avoidable and to deal with such attack, the

system administrators of the machines on which the protocol is run or the implementers of the protocol have to limit the number of attempts that a party can perform to authenticate or block a password after it is used for a certain number of failed attempts to authenticate. In an off-line attack, an adversary try to verify her guess for the password without running the protocol. A PAKE protocol has to be designed in a way such that an adversary can launch on-line dictionary attacks but it should be infeasible for her to verify her guess for the password off-line, without running the protocol.

### 1.1 Contribution

We have implemented a two-party AKE protocol in C programming language, using the cryptographic library of the OpenSSL project [49]. Following the general paradigm introduced by Cash et al. [25], we base our AKE protocol on a Weak Key Exchange (WKE) protocol and a Universally Composable PAKE protocol. Informally, a WKE protocol provides two parties with passwords that are individually unpredictable, while a PAKE protocol allows two parties to securely derive a key over an insecure channel using a low-entropy password. In particular, we have implemented:

1. in the Bounded Retrieval Model (BRM), a WKE protocol to output the password needed by the parties to run the PAKE protocol, similar but simpler than that one showed in [25], which uses less random bits than our protocol but does not seem to be efficiently implementable in practice;
2. the two-party PAKE protocol, showed secure in the Universal Composability (UC) framework in [1], which is more efficient than the Canetti et al.'s UC-PAKE protocol [23] used in [25].

Moreover, as a contribution, in Section 4.1 we formally adapt the security analysis of the WKE protocol of [25] to our WKE protocol.

In our model, before running the protocols, the two parties, Client and Server, have to agree on a shared large secret key $F$, a file of $n$ bits (some Gigabytes). They will use this file in the WKE protocol to compute the password for the PAKE protocol. Further, an active adversary $\mathcal{A}$ controls the channel between the parties and, before the parties run WKE and PAKE protocols, she can tamper the parties' machines and perform any efficient computation on the entire internal state of the parties, hence also the shared file $F$, but, as we work in the Bounded Retrieval Model, we assume that the adversary is communication bounded and can retrieve at most $\lambda$ bits of information from the internal state of Client and Server. In contrast, in our implementation Client can choose how many bits of the shared file $F$ she wants to use in the AKE protocol with Server.

Usually, a PAKE protocol deals with passwords with low entropy but in our setting the passwords output by the WKE protocol, and then used in the PAKE protocol, have high min-entropy from the adversary point of you. The important role of the PAKE protocol is to provide the client-to-server authentication: Client and Server will agree on a shared secret key after the PAKE protocol only if the passwords that they obtain at the end of the WKE protocol are equal. In fact, an adversary, who controls the channel and knows some information about the internal state of the parties and the shared file $F$, can adaptively correlate the passwords obtained by Client and Server at the end of the WKE protocol, without violate the individual unpredictability of the passwords. Several security models do not cover the realistic scenario in which the parties run the protocol with different but related passwords. For this reason, we use a UC-PAKE protocol, as in the UC framework there are not assumptions about the distribution of the passwords used by the parties who run the protocol. Canetti et al. [23] showed that UC-PAKE protocols cannot be achieved in the standard model and proposed a construction in the common reference string (CRS) model (used in the AKE protocol of [25]), in which the parties running the protocol agree, in a setup phase, to a shared short string. Clearly, it is simple to implement a CRS in the BRM as in this model the parties already share a large file and hence they should simply generate the CRS and store it as a part of their shared string but the construction in [23] is less efficient than other known constructions, developed in the Random Oracle (RO) and Ideal Cipher (IC) models. Therefore, we decide to implement the

UC-PAKE construction of Abdalla et al. [1], which is secure under the CDH intractability assumption and relies on the RO and IC models and is more efficient than the protocol of [23]. We deal with the controversial issues of implementing a RO and an IC with some concrete cryptographic hash functions and block cipher respectively [21], using the tools provided by the OpenSSL cryptographic library and following the standard choices showed in the RFC 5683 - Password-Authenticated Key Diffie-Hellman Exchange [18].

**OpenSSL project.** As stated on the OpenSSL project web page [49]: "The OpenSSL Project is a collaborative effort to develop a robust, commercial-grade, full-featured, and Open Source toolkit implementing the Secure Sockets Layer and Transport Layer Security protocols as well as a full-strength general purpose cryptography library". In this paper, we use only the OpenSSL cryptographic library, the last version `1.0.0e` released on September 2011, which "implements a wide range of cryptographic algorithms used in various Internet standards".

### 1.2 Related works

*Privacy amplification* and *authenticated key exchange* based on weak keys that have some entropy were studied in [55,48,52,30]. All the protocols showed therein are information-theoretic but they are not local, i.e. the parties need to use the whole secret key in their computation. Remind that, in our setting, Client can choose the number of bits of the shared password that the parties have to use when they run the protocol. Moreover, we will show at the end of Section 4.1 that the parties have to access a small portion of the shared file to achieve security against an adversary who can retrieve a very large amount of bits, even 99% of the shared file.

In the *exposure-resilient* cryptography [20,34,44] an adversary have almost complete access to the secret key, but she can only learn the bits of the key (which is short) and she cannot perform arbitrary computation on it. Also the protocols proven secure in this setting are not local. To deal with key exposure, some stateful schemes have been introduced in which the key evolves during the time and an adversary can learn the whole key but the protocols update their secret key at each round. In this setting, *forward secure* schemes [5,11,22] provide security for all the rounds occurred before that the adversary tampers the system and learns the secret key but, once this happens, there is no more security for the following rounds.

*Key-insulated* cryptography [31,32] guarantees security for past and future rounds but it needs a master key, used by the parties to update their secret keys at every round and stored by a party that the adversary cannot corrupt. As an extension of such models, (two-party) *intrusion-resilient* schemes [42,29] consider an adversary who can compromises both the parties but not in the same round.

In 2006, Dziembowski [35] constructed the first secure intrusion-resilient authenticated key exchange, in the Bounded-Storage Model, using random oracles. Influenced by the work of Boyen et al. [14], in which the authors study the AKE protocol which uses biometric data and introduce "the idea of using low entropy intermediate keys as input of a PAKE protocol", Cash et al. [25] showed a general construction of an intrusion-resilient AKE protocol secure against static adversary and achieved an instantiation of such paradigm without random oracles, in the BRM. In our implementation, we follow the general paradigm of [25] (implicitly used also in [35]), in which an AKE protocol is realized by performing a WKE protocol, which provides the parties with individually unpredictable passwords, and then a PAKE protocol, which uses such passwords as input and outputs a shared secret key only when both parties run the protocol with the same passwords.

In 1992 Bellovin and Merrit [12] introduced the notion of PAKE but only in 2000 Bellare et al. [6] and Boyko et al. [15] provided the first formalization of models and security of PAKE protocols. Both these works studies the Encrypted Key Exchange (EKE) protocol introduced in [12], a DH Key Exchange protocol in which the public values of the parties are encrypted before to be sent from a party to the other, using the shared password as the key for the encryption and decryption operations. Bresson et al. [16] showed a formal security proof for a PAKE protocol in the model of [6], which extends the framework of Bellare and Rogaway [7,8],

relying on the RO model and the IC model, while Boyko et al. [15] provided such a proof relying only on the RO model, using the framework of Shoup [53]. There are also some works [38,45,46] that provide security proofs for PAKE protocol in the standard model, relying on the DDH assumption, but such protocols are not efficiently implementable. A more recent version of the EKE protocol is the AuthA protocol[1] [9], modeled in [16] by the One-Encryption Key Exchange (OEKE) protocol, in which only the public value of the server is encrypted before to be sent to the client, while the first flow from the client to the server is sent in clear, without any encryption. In [17], Bresson et al. showed a construction of the AuthA protocol which does not rely on the IC model but uses a full-domain hash scheme, that is provable in the RO model, to perform the encryption process. Instead of using a block cipher, they encrypt the server's public value by multiplying it by a full-domain hash of the shared password.

In [23] Canetti at al. pointed out that the security models of [6] and [15] do not cover the realistic scenarios in which the parties run the protocol with "different but possibly correlated passwords" and then they introduced a new security model in the UC framework for PAKE protocol, in which there are "no assumptions on the distribution on the passwords" used by the parties who run the protocol. Moreover, in [23], the authors showed a PAKE protocol and proved its security in their new model, relying on standard number-theoretic assumptions, against static adversaries, in the CRS model. They also showed that it is impossible to achieve UC-PAKE in the standard model. Their protocol is based on those ones of Katz et al. [45] and Gennaro and Lindell [37] but it is not as efficient as some protocols proven secure in the models of [6] and [15], e.g. [16,2,45,47]. Abdalla et al. [1] showed that the efficient PAKE protocol studied in [16] and proven secure in the model of [6] is also secure in the UC framework of Canetti et al. [23]. Indeed, in [1] the authors proven the security of the UC-PAKE protocol against adaptive adversaries under the CDH intractability assumption, in the RO model and the IC model.

## 2 Preliminaries

**Definition 1 (Min-entropy).** *Given a random variable $X$, the min-entropy of $X$ is* $\mathbf{H}_\infty(X) \stackrel{\mathsf{def}}{=} \min_x(-\log(\boldsymbol{Pr}[X = x])) = -\log(\max_x(\boldsymbol{Pr}[X = x]))$.

For cryptographic applications Dodis et al. [33] introduced the notion of *average conditional min-entropy*:

**Definition 2 (Average conditional min-entropy).** *Given two random variables $X, Y$, the average conditional min-entropy of $X$ given $Y$ is* $\tilde{\mathbf{H}}_\infty(X|Y) \stackrel{\mathsf{def}}{=} -\log(\mathbb{E}_{y \leftarrow Y}(\max_x(\boldsymbol{Pr}[X = x|Y = y])))$.

Let $\mathcal{G}(1^k)$ be a group sampling algorithms which outputs a tuple $(\mathbb{G}, p, g)$, where $\mathbb{G}$ is a cyclic group of order $p$ and $g \in \mathbb{G}$ is a generator of $\mathbb{G}$ (in practice, prime order groups are desirable).

**Definition 3 (Computational Diffie-Hellman (CDH) problem).** *Let $x, y$ uniformly distributed over $\mathbb{Z}_p$ and $(\mathbb{G}, p, g) \leftarrow \mathcal{G}(1^k)$. The CDH problem is hard for $\mathbb{G}$ if for every adversary $\mathcal{A}$ there exists a negligible function $negl(\cdot)$ such that* $\boldsymbol{Pr}[\mathcal{A}(\mathbb{G}, p, g, g^x, g^y) = g^{xy}] \leq negl(k)$.

**Definition 4 (Decisional Diffie-Hellman (DDH) problem).** *Let $x, y, z$ uniformly distributed over $\mathbb{Z}_p$ and $(\mathbb{G}, p, g) \leftarrow \mathcal{G}(1^k)$. The DDH problem is hard for $\mathbb{G}$ if for every adversary $\mathcal{A}$ there exists a negligible function $negl(\cdot)$ such that* $|\boldsymbol{Pr}[\mathcal{A}(\mathbb{G}, p, g, g^x, g^y, g^{xy}) = 1] - \boldsymbol{Pr}[\mathcal{A}(\mathbb{G}, p, g, g^x, g^y, g^z) = 1]| \leq negl(k)$.

In other words, given the tuple $(\mathbb{G}, p, g)$ and the randomly chosen $g^x$ and $g^y$, the CDH problem is to compute the Diffie-Hellman value $g^{xy}$ while the DDH problem is to distinguish the Diffie-Hellman value $g^{xy}$ from a randomly chosen group element $g^z$.

[1] Considered for standardization by the IEEE P1363 Standard working group on public-key cryptography [41]

**Bounded-Retrieval Model.** The Bounded-Retrieval Model (BRM) was introduced independently by Dziembowski [35] and Di Crescenzo et al. [27]. In this model, the parties share a huge (several Gygabytes) random secret key $K$ and an adversary is allowed to perform any computation on it but the quantity of information that she can retrieve is bounded, i.e. it is less than $|K|$ but it can even consist of some Gygabytes. This model captures the setting in which one wants to protect secret keys and to preserve security of cryptographic schemes on a computer infected (for a limited amount of time) by a malware, like a virus, which can communicate only a bounded amount of information to a malicious user, after performing any efficient computation on the entire internal state of the infected machine.

More precisely, an adversary can apply any input-shrinking function $\Lambda$ to $K$ and retrieve the result and after this step it is assumed that the parties are able to remove the malware program from their machine and then they use only a small portion of their shared secret key $K$ to perform some cryptographic scheme. Informally, a scheme is secure in the BRM if the secret key $K$ remains indistinguishable from random to an adversary even conditioned on the knowledge of the retrieved information $\Lambda(K)$.

The assumption that an adversary cannot retrieve more than some Gygabytes is quite practical as an honest user should be able to notice whether her machine is transmitting such huge amount of information to the external world, independently by her actions.

In the past years, authentication schemes [35,27,25] and secret-sharing [36] were proven secure in the BRM. Recently, Alwen et al. [4,3] showed how to build the first public-key primitives (identification schemes, signatures, authenticated key agreement and encryption) in the BRM based on a variety of assumptions.

**Universally Composable Security.** In [19] Canetti introduced the Universally Composable (UC) framework in which security is based on an ideal functionality $\mathcal{F}$, which can be viewed as a trusted party that interacts with a group of players to compute some given function $f$. The functionality $\mathcal{F}$ receives the input of each party, computes $f$ on these values and gives back the output to the players. Hence, in this setting an attacker can only retrieve the data of the corrupted players but security is guaranteed for the honest parties.

Informally, a protocol proven secure in the UC framework remains secure even when used in arbitrary "environments", i.e. the behavior of a protocol which implements an "ideal functionality", under an adversarial attack, is not distinguishable from the behavior of a simulated attack on the "ideal functionality", by an "environment" that controls the parties who run the protocol, choosing inputs and observing outputs, during an attack. By this property, an "ideal functionality" can be used in any construction and then it can be replaced by a UC secure protocol, guaranteeing that the security analysis of the construction still holds.

In the UC framework there may exist several copies of the ideal functionality $\mathcal{F}$ running at the same time. Each copy has a unique session identifier (SID). The original UC theorem analyzes the security of a single protocol, but it does not consider the case in which several protocols share state and randomness. In [24] Canetti and Rabin introduced the UC with joint state, which guarantees security in case different protocols share some common state. Informally, they define a multisession extension $\mathcal{F}^*$ of $\mathcal{F}$, which runs several executions of $\mathcal{F}$. Each copy of $\mathcal{F}$ is identified by a sub-session identifier (SSID).

## 3 Implementation

In this section, we describe the Authenticated Key Exchange protocol that we have implemented in `C` programming language, using the cryptographic library of the OpenSSL project. Our AKE is based on a WKE protocol and a UC-PAKE protocol, we show them in the next subsections.

### 3.1 Weak Key Exchange

In a WKE protocol (see Figure 1), two parties, Client and Server, sharing a source (a large file), simply choose

**Setup**:

- $F \in_R \{0,1\}^n$: a large file (some Gigabytes) shared between Client and Server.
- $\texttt{DirProd} : \Sigma^n \times [n]^t \rightarrow \Sigma^t$, a function which takes a source $y = (y[1], \ldots, y[n]) \in \Sigma^n$ and indexes $(i_1, \ldots, i_t) \in [n]^t$ and outputs $(y[i_1], \ldots, y[i_t])$, the elements of the source at the provided indexes.

**Protocol**:

1. Client and Server choose random $I_C, I_S \in [n]^t$, respectively.
2. Client sends $I_C$ to Server. Server receives a value $I'_C$ and computes $Pwd_S = \texttt{DirProd}(F, I'_C)||\texttt{DirProd}(F, I_S)$. Server outputs $Pwd_S$ as her password for the PAKE protocol.
3. Server sends $I_S$ to Client. Client receives a value $I'_S$ and computes $Pwd_C = \texttt{DirProd}(F, I_C)||\texttt{DirProd}(F, I'_S)$. Client outputs $Pwd_C$ as her password for the PAKE protocol.

**Fig. 1.** Weak Key Exchange Protocol

a set of indexes from the source, send it to each other and compute a password as the concatenation of the bits of the source at the provided (ordered) indexes, first the Client's indexes and then the Server's ones. The WKE construction of Cash et al. [25] makes use of an averaging sampler [10], which samples a small amount of bits from a large source in such a way that the min-entropy rate of the sampled bits is nearly the same of the large source. In their protocol, Cash et al. uses an explicit construction of averaging sampler showed by Vadhan [54], which uses less random bits than the random choice. However, an implementation of such sampler does not seem to be practical. In fact, it is not clear how to calculate the hidden constants of the Vadhan's result, as it is based on properties of expander graphs and their second eigenvalue for which only asymptotic values are known, and thus it is not possible to calculate the exact amount of random bits needed.

In our construction, to sample some bits from the large shared source, instead of using random walks on expander graphs, we use the random choice and then, to extract the selected bits from the source, we use the simple function Direct Product (implicitly used also in [25]), which takes as input a source and a set of indexes and outputs the elements of the source at the provided indexes. A result of Alwen et al. (Lemma A.3 in [4]) shows that, choosing appropriate parameters, the bits output by the function Direct Product have good min-entropy. It means that the shared passwords obtained by the parties at the end of the WKE protocol are individually unpredictable for the adversary, they will be equal if the adversary is passive while they could be different and correlated if the adversary performs some action on the messages exchanged between the parties, e.g. the adversary can block $I_C$ or $I_S$ and send a different, chosen by her, $I'_C$ or $I'_S$.

Our WKE protocol is not forward secure, i.e. it does not provide security for passwords in case of future adversarial break in, as when an adversary compromises one of the party's machine, she could be able to retrieve (without violating the retrieval bound) the bits of the shared file which were used to compute the passwords in a previous execution of the WKE protocol. We show the analysis of our WKE protocol in Section 4.1.

**Implementation details.** To use the Direct Product function, the two parties running the WKE protocol have to choose uniformly at random the set of indexes needed by this function and then they should send to each other their own indexes. As the size of the shared file could consist of several Gigabytes, the value of each index, which represents the position of a bit in the source, could be a large integer, thus we have to use the `C` "long long int" type to store it, i.e. 8 bytes. It means that, to sample $t$ bits from the source, the parties have to use $64 * t$ random bits and they have to send these bits to each other. To make our protocol more efficient in terms of random bits and communication complexity, we use a hash function to produce the indexes needed by the Direct Product function. When Client runs the protocol, she chooses a cryptographic hash function $\mathcal{H}$ (for example SHA-1) and sends it to Server. After this step, Client and Server have to choose respectively only a

small random seed $s$, e.g. 4 bytes, and they can calculate their $t$ indexes $idx_1, \ldots, idx_t$ as follows:

$$idx_i = ((\mathcal{H}(i||s) \bmod 2^{64}) \bmod fileSizeBits) + 1, \tag{1}$$

i.e. as the decimal value of the least significant 8 bytes (so it can represent an integer between 0 and $2^{64} - 1$) of the digest output by the hash function applied to the number $i$ concatenated to the seed $s$, modulo *fileSizeBits*, the size in bits of the shared file, plus 1. Hence, $\forall\, i \in [t]$, it holds that $idx_i \in [fileSizeBits]$, i.e. every index represents the position of a bit of the source. In this way, Client and Server uses only few random bits and they do not need to send all the indexes to the other party, as they can simply send their own seed and then they can calculate the indexes of the other party by their own, applying the hash function $\mathcal{H}$ to the seed that they received. Our solution is very flexible, as Client can choose to use even all the bits of the shared source to create the password, tolerating in this way the maximum amount of leakage, with constant communication complexity and constant number of random bits used by each party, at the cost, of course, to rely on the RO model. However, we would like to note that in our implementation of the AKE protocol we need also a UC-PAKE protocol which already relies on RO, as it is impossible to achieve UC-PAKE in the standard model [23].

### 3.2 Password-based Authenticated Key Exchange

In our AKE protocol, we implement the two-party UC-PAKE protocol, proven secure in the RO and IC models, under the CDH assumption, by Abdalla et al. in [1]. In this protocol (see Figure 2), the parties, Client and Server, share a password and perform an one-flow encrypted Diffie-Hellman key exchange with one-side authentication.

**Setup**:

- Client and Server share a sub-session identifier *ssid* and a password *pwd*.
- A finite cyclic group $\mathbb{G} = \langle g \rangle$ of order a $\ell$-bit prime number $p$, where the operation is denoted multiplicatively.
- A block cipher: $(\mathcal{E}_k, \mathcal{D}_k)$.
- Two hash functions: $\mathcal{H}_0 : \{0,1\}^* \to \{0,1\}^{h_0}$ and $\mathcal{H}_1 : \{0,1\}^* \to \{0,1\}^{h_1}$.
- The PKCS5-PBKDF2-HMAC-SHA1 pseudo-random function.

**Protocol**:

| Client **C** | | Server **S** |
|---|---|---|
| $priv_{\mathbf{C}} \xleftarrow{R} [p-1]$ | | $priv_{\mathbf{S}} \xleftarrow{R} [p-1]$ |
| $pub_{\mathbf{C}} := g^{priv_{\mathbf{C}}}$ | $\xrightarrow{\mathbf{C}, pub_{\mathbf{C}}}$ | $pub_{\mathbf{S}} := g^{priv_{\mathbf{S}}}$ |
| $pub_{\mathbf{S}} := \mathcal{D}_{\texttt{PKCS5-PBKDF2-HMAC-SHA1}(ssid\|\|pwd)}(pub^*_{\mathbf{S}})$ | $\xleftarrow{\mathbf{S}, pub^*_{\mathbf{S}}}$ | $pub^*_{\mathbf{S}} := \mathcal{E}_{\texttt{PKCS5-PBKDF2-HMAC-SHA1}(ssid\|\|pwd)}(pub_{\mathbf{S}})$ |
| $K_{\mathbf{C}} := pub_{\mathbf{S}}^{priv_{\mathbf{C}}}$ | | $K_{\mathbf{S}} := pub_{\mathbf{C}}^{priv_{\mathbf{S}}}$ |
| $Auth := \mathcal{H}_1(ssid\|\|\mathbf{C}\|\|\mathbf{S}\|\|pub_{\mathbf{C}}\|\|pub_{\mathbf{S}}\|\|K_{\mathbf{C}})$ | $\xrightarrow{Auth}$ | if $(Auth = \mathcal{H}_1(ssid\|\|\mathbf{C}\|\|\mathbf{S}\|\|pub_{\mathbf{C}}\|\|pub_{\mathbf{S}}\|\|K_{\mathbf{S}}))$ |
| $sk_{\mathbf{C}} := \mathcal{H}_0(ssid\|\|\mathbf{C}\|\|\mathbf{S}\|\|pub_{\mathbf{C}}\|\|pub_{\mathbf{S}}\|\|K_{\mathbf{C}})$ | | then $sk_{\mathbf{S}} := \mathcal{H}_0(ssid\|\|\mathbf{C}\|\|\mathbf{S}\|\|pub_{\mathbf{C}}\|\|pub_{\mathbf{S}}\|\|K_{\mathbf{S}})$ |
| `completed` | | `completed` |
| | | else `error` |

**Fig. 2.** Password-Authenticated Key Exchange Protocol

To run the protocol, the parties have to agree on some parameters: a block cipher: $(\mathcal{E}_k, \mathcal{D}_k)$, two hash functions: $\mathcal{H}_0 : \{0,1\}^* \to \{0,1\}^{h_0}$ and $\mathcal{H}_1 : \{0,1\}^* \to \{0,1\}^{h_1}$ and a finite cyclic group $\mathbb{G} = \langle g \rangle$ of

order a prime number $p$, where the operation is denoted multiplicatively. Moreover, Client and Server share a sub-session identifier, which they can compute exchanging nonces, and a password. In our setting, before running the protocol Client chooses the parameters and sends them to Server. The protocol performs three flows. Client chooses a private DH random exponent $priv_{\mathbf{C}} \in [p-1]$, calculates the corresponding public DH value $pub_{\mathbf{C}} := g^{priv_{\mathbf{C}}}$ and sends it to Server, together with her name **C**. Similarly, Server chooses a private DH random exponent $priv_{\mathbf{S}} \in [p-1]$ and calculates the corresponding public DH value $pub_{\mathbf{S}} := g^{priv_{\mathbf{S}}}$ but before to send it to Client, together with her name **S**, Server encrypts it, using the password $pwd$ and the sub-session identifier $ssid$ to compute a key for the block cipher. Such key can be computed by the Public-Key Cryptography Standards (PKCS) #5 Password-Based Key Derivation Function 2 (PBKDF2) with the Hash-based Message Authentication Code (HMAC) Secure Hash Algorithm (SHA-1) pseudo-random function [43]. Server computes the DH secret key $g^{priv_{\mathbf{C}} priv_{\mathbf{S}}}$ as $K_{\mathbf{S}} := pub_{\mathbf{C}}^{priv_{\mathbf{S}}}$. To decrypt the public value of Server, Client apply the same PKCS5-PBKDF2-HMAC-SHA-1 function used by Server to $pwd$ and $ssid$ to obtain the same key for the block cipher. Then Client computes the DH secret key $K_{\mathbf{C}} := pub_{\mathbf{S}}^{priv_{\mathbf{C}}}$, the secret key $sk_{\mathbf{C}} := \mathcal{H}_0(ssid||\mathbf{C}||\mathbf{S}||pub_{\mathbf{C}}||pub_{\mathbf{S}}||K_{\mathbf{C}})$ and an authentication tag $Auth := \mathcal{H}_1(ssid||\mathbf{C}||\mathbf{S}||pub_{\mathbf{C}}||pub_{\mathbf{S}}||K_{\mathbf{C}}))$. Client sends this tag to Server. Then Server computes $\mathcal{H}_1(ssid||\mathbf{C}||\mathbf{S}||pub_{\mathbf{C}}||pub_{\mathbf{S}}||K_{\mathbf{S}}))$, if it is equal to $Auth$ then Server computes the secret key $sk_{\mathbf{S}} := \mathcal{H}_0(ssid||\mathbf{C}||\mathbf{S}||pub_{\mathbf{C}}||pub_{\mathbf{S}}||K_{\mathbf{S}})$ and the protocol succeeds in providing Client and Server with a secret key, otherwise Server produce an error message instead of the secret key. We discuss the analysis of this UC-PAKE protocol in Section 4.2.

### 3.3 Authenticated Key Exchange from Weak Key Exchange and Password-based Authenticated Key Exchange

Following the theoretical work of [25], in our implementation of the AKE protocol (see Figure 3), two parties,

**Setup**:

- Client and Server share a large file.
- WKE: A Weak Key Exchange protocol.
- UC-PAKE: A Universally Composable Password-based Authenticated Key Exchange protocol.

**Protocol**:

1. Client chooses the parameters of the protocols and sends them to Server.
2. Client and Server run the WKE protocol, obtaining the passwords $Pwd_C$ and $Pwd_S$, respectively.
3. Client and Server exchange nonces, which are concatenated to form a sub-session identifier *ssid*.
4. Client and Server run the UC-PAKE protocol with sub-session identifier *ssid*, using the passwords $Pwd_C$ and $Pwd_S$ as their inputs, respectively. Client obtains secret keys $sk_C$ and Server obtains $sk_S$ secret key or an error message.

**Fig. 3.** Authenticated Key Exchange Protocol

sharing a large file, run a WKE protocol to derive a shared password, with high min-entropy from the adversary point of view, and then they use this password in a UC-PAKE protocol to compute a shared secret key and to provide client-to-server authentication.

Before running the WKE protocol, Client selects the parameters of the protocols and sends them to Server, in one round. Our WKE construction, based on that one in [25], is a two round protocol and the UC-PAKE protocol that we implemented from [1] is a three round protocol. Two more rounds are needed to exchange

nonce, for a total of 8 rounds. However, we have moved the first round and the exchange of nonce in the WKE protocol, using three rounds less, for a total of 5 rounds. The AKE protocol construction showed in [25], uses 9 rounds but achieves two-side authentication. In our implementation, only client-to-server authentication is performed, as only Server receives an error message in case the parties do not meet the condition to share the same key. To achieve mutual authentication, we have to add just one round in the UC-PAKE protocol but then we should carefully check if the security proofs given in [1] and [16] still hold. However, as stated in [1]: "client-authentication is usually enough in most cases and often results in more efficient protocols". We show the analysis of our AKE protocol in Section 4.3.

# 4 Analysis

## 4.1 Weak Key Exchange

In this section we adapt the security analysis of the WKE protocol of [25] to our WKE protocol (see Section 3.1). We do not introduce a formal security model as it would be equal to that one introduced in [25] but an interested reader can easily check that the result obtained in this section make our WKE protocol straightforwardly secure in that model.

In our construction of the WKE protocol, we use the Direct Product function $\texttt{DirProd} : \Sigma^n \times [n]^t \rightarrow \Sigma^t$, which takes as input a source $y = (y[1], \ldots, y[n]) \in \Sigma^n$ and a set of indexes $(i_1, \ldots, i_t) \in [n]^t$ and outputs $(y[i_1], \ldots, y[i_t])$, the elements of the source at the provided indexes.
In this section, we show that, choosing appropriate parameters, the elements output by the Direct Product function have a good min-entropy. We need the following results:

**Lemma 1 (Lemma 1 from [25]).** *Let $X$ and $Y$ be any two (correlated) random variables. Suppose that $\mathbf{H}_\infty(X) \geq n$, and $Y$ takes values in $\{0,1\}^r$. Then for every $\epsilon > 0$, with probability at least $1 - \epsilon$ over $y \leftarrow Y$, $\mathbf{H}_\infty(X|Y = y) \geq (n - r - \log(1/\epsilon))$.*

**Lemma 2 (Lemma A.3 from [4]).** *Let $\Sigma$ be an alphabet of size $q$. Let $X$ be a random variable over $\Sigma^n$ and $\mathcal{E}_1$ be an arbitrary experiment. Define $\mathcal{E}_2$ to be the experiment where, at the conclusion of $\mathcal{E}_1$, a uniformly random $R \in [n]^t$ is chosen and given to the predictor. Then for any $c > 0$, if $\tilde{\mathbf{H}}_\infty(X|\mathcal{E}_1) \geq \frac{2cn}{t}(\log_2(q) + \log_2(n)) + 3c + 5$ then $\tilde{\mathbf{H}}_\infty(\texttt{DirProd}(X, R)|\mathcal{E}_2) > c$.*

**Lemma 3 (Lemma 2.2 from [33]).** *Let $A, B$ be random variables. For any $\delta > 0$, $\mathbf{H}_\infty(A|B = b)$ is at least $\tilde{\mathbf{H}}_\infty(A|B) - \log_2(1/\delta)$ with probability at least $1 - \delta$ over the choice of $b$.*

Our result can be stated as follows:

**Lemma 4.** *Let $\texttt{DirProd} : \Sigma^n \times [n]^t \rightarrow \Sigma^t$ be as above and let $F$ be a uniformly random string in $\{0,1\}^n$. Then, for every $0 < \beta, \epsilon < 1$, $t < n$, $\lambda < n$ and $k > 0$,*

$$\text{if } n - \lambda - \beta \geq \frac{2kn}{t}(1 + \log_2(n)) + 3k + 5 \text{ then } \mathbf{H}_\infty(\texttt{DirProd}(F, I_C)|(I_C, view_{\mathcal{A}}) = (\mathcal{I}, \mathcal{V})) > k - \epsilon,$$

*with probability $1 - 2^{-\epsilon}$, over the choice of $\mathcal{I} \in [n]^t$ and $\mathcal{V} \in \{0,1\}^\lambda$.*

*Proof.* We assume that the file $F$ is generated uniformly at random by Server and given to Client in a setup phase. As $F$ is uniformly random it has min-entropy equals to its number of bits, i.e. $\mathbf{H}_\infty(F) = n$. Before the beginning of the WKE protocol, the adversary can compromise the machines of Client and Server, perform any efficient computation on their internal states and the shared file $F$ but, as we work in the BRM, she can retrieve at most $\lambda$ bits of information. We can think of this $\lambda$ bits as the output of a circuit chosen by the adversary and

applied to the entire internal state of Client and Server, so also $I_C, I_S$ and $F$ (cf. Figure 1). Our setting covers also the adaptive case, in which the adversary chooses adaptively several circuits and applies them sequentially once she intruded into the machines, the only restriction is that the total amount of bits output by the circuits and then retrieved by the adversary is at most $\lambda$. Hence, by Lemma 1, with probability $1 - 2^{-\beta}$ taken over the distribution of the adversary view $view_{\mathcal{A}} \leftarrow \{0,1\}^{\lambda}$, i.e. the vector of the $\lambda$ bits retrieved by the adversary from Client and Server, it holds that $\mathbf{H}_{\infty}(F|view_{\mathcal{A}}) \geq n - \lambda - \beta$, for any $\beta$. It means that from the adversary's point of view the min-entropy of $F$ decreases at most of $\lambda + \beta$, after the adversary learns $view_{\mathcal{A}}$, the vector of the retrieved $\lambda$ bits, with probability $1 - 2^{-\beta}$.

In our setting we can think of the experiment $\mathcal{E}_1$ of Lemma 2 as the random variable $view_{\mathcal{A}}$, hence applying Lemma 2, we obtain that for any $k > 0$,

$$\text{if } \tilde{\mathbf{H}}_{\infty}(F|view_{\mathcal{A}}) \geq \frac{2kn}{t}(1 + \log_2(n)) + 3k + 5 \text{ then } \tilde{\mathbf{H}}_{\infty}(\mathtt{DirProd}(F, I)|(I, view_{\mathcal{A}})) > k, \tag{2}$$

where $I \in_R [n]^t$.[2]
As $F$ is uniformly random and $view_{\mathcal{A}}$ is a vector of $\lambda$ bits, we have that, from the adversary's point of view, $\tilde{\mathbf{H}}_{\infty}(F|view_{\mathcal{A}}) = \mathbf{H}_{\infty}(F|view_{\mathcal{A}}) \geq n - \lambda - \beta$. Then, fixing a value for $k$, equation (2) implies that

$$\text{if } n - \lambda - \beta \geq \frac{2kn}{t}(1 + \log_2(n)) + 3k + 5 \text{ then } \tilde{\mathbf{H}}_{\infty}(\mathtt{DirProd}(F, I_C)|(I_C, view_{\mathcal{A}})) > k. \tag{3}$$

Now, by fixing $\delta = 2^{-\epsilon}$ and applying Lemma 3, we obtain that

$$\mathbf{H}_{\infty}(\mathtt{DirProd}(F, I_C)|(I_C, view_{\mathcal{A}}) = (\mathcal{I}, \mathcal{V})) \geq \tilde{\mathbf{H}}_{\infty}(\mathtt{DirProd}(F, I_C)|(I_C, view_{\mathcal{A}})) - \epsilon \tag{4}$$

with probability $1 - 2^{-\epsilon}$, over the choice of $\mathcal{I} \in [n]^t$ and $\mathcal{V} \in \{0,1\}^{\lambda}$.
Then, combining (3) and (4), we have that

$$\text{if } n - \lambda - \beta \geq \frac{2kn}{t}(1 + \log_2(n)) + 3k + 5 \text{ then } \mathbf{H}_{\infty}(\mathtt{DirProd}(F, I_C)|(I_C, view_{\mathcal{A}}) = (\mathcal{I}, \mathcal{V})) > k - \epsilon, \tag{5}$$

with probability $1 - 2^{-\epsilon}$, over the choice of $\mathcal{I} \in [n]^t$ and $\mathcal{V} \in \{0,1\}^{\lambda}$. □

In our implementation, we fix $k = 2\epsilon$, as in this way $\epsilon$ represents the lower bound of the min-entropy in equation (5) and it is the only security parameter of the protocol. Hence, we finally obtain:

**Corollary 1.** *Let* $\mathtt{DirProd} : \Sigma^n \times [n]^t \to \Sigma^t$ *be as above and let* $F$ *be a uniformly random string in* $\{0,1\}^n$. *Then, for every* $0 < \beta, \epsilon < 1$, $t < n$ *and* $\lambda < n$,

$$\textit{if } n - \lambda - \beta \geq \frac{4\epsilon n}{t}(1 + \log_2(n)) + 6\epsilon + 5 \tag{6}$$

$$\textit{then } \mathbf{H}_{\infty}(\mathtt{DirProd}(F, I_C)|(I_C, view_{\mathcal{A}}) = (\mathcal{I}, \mathcal{V})) > \epsilon, \tag{7}$$

*with probability* $1 - 2^{-\epsilon}$, *over the choice of* $\mathcal{I} \in [n]^t$ *and* $\mathcal{V} \in \{0,1\}^{\lambda}$.

Therefore, even conditioned on the adversary's view (the $\lambda$ bits retrieved from the internal state of the parties by the adversary) and the set of $t$ indexes accessed in the file $F$ by Client, if equation (6) holds, $Pwd_C$ has min-entropy greater than $\epsilon$ (with probability $1 - 2^{-\epsilon}$), for any $I'_S$ chosen by the adversary (cf. Figure 1). A similar

[2] Note that the adversary can learn the parties' seeds observing the channel and then she can simply calculate the corresponding indexes by applying the formula in equation (1). Moreover, she can easily retrieve the length of the shared file when she compromises the parties' machines before they run the protocol.

argument applies to $Pwd_S$, therefore we conclude that, choosing appropriate parameters, the elements output by the Direct Product function have a good min-entropy from an adversary's point of view, even after retrieving $\lambda$ bits of information from the internal state of Client and Server (before the execution of the WKE protocol). It means that the shared passwords obtained by the parties at the end of the WKE protocol are individually unpredictable for the adversary, they will be equal if the adversary is passive while they could be different and correlated if the adversary performs some action on the messages exchanged between the parties.

**Concrete parameters and experimental evaluation.** Fixing $\epsilon$ and $\beta$, we can choose the level of security that we want to achieve and then we can study the relation between $\lambda$, the number of bits retrieved by the adversary from the file $F$, and $t$, the number of bits of the file $F$ that Client and Server use to compute their password. We simply set $\beta = 80$ to have that with very large probability $1-2^{-80}$, it holds that $\mathbf{H}_\infty(F|view_\mathcal{A}) \geq n - \lambda - 80$. This choice does not affect the efficiency of the protocol as $n$ is much greater than 80. In our implementation, by default, we set $\epsilon = 30$, obtaining that with probability $1 - 2^{-30}$ the min-entropy of $Pwd_C$ and $Pwd_S$ is greater than 30 (after the WKE protocol terminates and even if the adversary has retrieved $\lambda$ bits from Client and Server and knows $I_C$ and $I_S$), if $n - \lambda \geq \frac{120n}{t}(1+\log_2(n))+265$. However, Client can choose to set $\epsilon$ to any value greater than 30 when she runs the protocol. We note that the value $k$, which represents a lower bound of the average conditional min-entropy $\tilde{\mathbf{H}}_\infty(\mathtt{DirProd}(F,I)|(I,view_\mathcal{A}))$, is fixed to be equal to $2\epsilon$, as in this way equation (7) holds and $\epsilon$ is the only security parameter of the protocol (cf. Corollary 1).

Table 1 describes the values of $t$ evaluated in the standard setting $\epsilon = 30$ and in an higher level of security setting in which $\epsilon = 80$ and the running time of the AKE protocol which is evaluated experimentally on an Intel(R) Core(TM) i5-2410M CPU @ 2.30GHz, with 4GB of RAM, under the 64-bits version of Ubuntu 11.04.

| | $\epsilon = 30$ | | | | | | | | | $\epsilon = 80$ | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | $\lambda = 90\%$ of $n$ | | | $\lambda = 95\%$ of $n$ | | | $\lambda = 99\%$ of $n$ | | | $\lambda = 90\%$ of $n$ | | | $\lambda = 95\%$ of $n$ | | | $\lambda = 99\%$ of $n$ | | |
| **n** | **t** | **WKE** | **PAKE** | **t** | **WKE** | **PAKE** | **t** | **WKE** | **PAKE** | **t** | **WKE** | **PAKE** | **t** | **WKE** | **PAKE** | **t** | **WKE** | **PAKE** |
| (GB) | (KB) | (sec) | (sec) | (KB) | (sec) | (sec) | (KB) | (sec) | (sec) | (KB) | (sec) | (sec) | (KB) | (sec) | (sec) | (KB) | (sec) | (sec) |
| 1 | 4,98 | 51 | 2 | 9,96 | 53 | 3 | 49,80 | 49 | 16 | 13,28 | 52 | 4 | 26,56 | 49 | 9 | 132,81 | 49 | 44 |
| 5 | 5,32 | 198 | 2 | 10,64 | 202 | 3 | 53,21 | 213 | 18 | 14,19 | 211 | 5 | 28,38 | 214 | 9 | 141,88 | 233 | 47 |
| 10 | 5,47 | 404 | 2 | 10,93 | 415 | 4 | 54,67 | 437 | 18 | 14,58 | 421 | 5 | 29,16 | 417 | 10 | 145,79 | 444 | 48 |
| 15 | 5,55 | 582 | 2 | 11,11 | 654 | 4 | 55,53 | 675 | 18 | 14,81 | 655 | 5 | 29,61 | 658 | 10 | 148,07 | 707 | 49 |
| 20 | 5,61 | 1251 | 2 | 11,23 | 952 | 4 | 56,14 | 982 | 19 | 14,97 | 957 | 5 | 29,94 | 973 | 10 | 149,7 | 1053 | 50 |
| 25 | 5,66 | 2561 | 2 | 11,32 | 1129 | 4 | 56,61 | 1225 | 19 | 15,1 | 1226 | 5 | 30,19 | 1240 | 10 | 150,95 | 1325 | 51 |
| 30 | 5,7 | 2409 | 2 | 11,4 | 1209 | 4 | 56,99 | 1354 | 19 | 15,2 | 1272 | 5 | 30,4 | 1345 | 10 | 151,98 | 1438 | 51 |
| 35 | 5,73 | 2886 | 2 | 11,46 | 1750 | 4 | 57,32 | 1880 | 19 | 15,28 | 1590 | 5 | 30,57 | 1837 | 10 | 152,85 | 1951 | 51 |
| 40 | 5,76 | 2982 | 2 | 11,52 | 2568 | 4 | 57,6 | 2106 | 20 | 15,36 | 1786 | 5 | 30,72 | 2092 | 10 | 153,6 | 2124 | 52 |
| 45 | 5,78 | 2809 | 2 | 11,57 | 3355 | 4 | 57,85 | 2109 | 20 | 15,43 | 1908 | 5 | 30,85 | 2069 | 10 | 154,27 | 2154 | 52 |
| 50 | 5,81 | 2916 | 2 | 11,61 | 4564 | 4 | 58,07 | 2427 | 20 | 15,49 | 2586 | 5 | 30,97 | 2360 | 11 | 154,86 | 2445 | 52 |

**Table 1.** Values of $t$ and running time of the WKE and PAKE protocols, in the standard setting $\epsilon = 30$ and in an higher level of security setting $\epsilon = 80$, for file size $1GB \leq n \leq 50GB$ and leakage $90\%$ of $n \leq \lambda \leq 99\%$ of $n$.

From the analysis of Table 1, we can conclude that in our AKE protocol implementation it is possible to achieve a very high leakage resilience percentage still accessing a small portion of the shared file, even for

$\epsilon = 80$, as Client and Server need to use only less than 310 KB[3] over the several Gigabytes of the shared file $F$ to securely run the AKE protocol against an adversary who can retrieve even up to 99% of the file, as also showed in Figure 4.

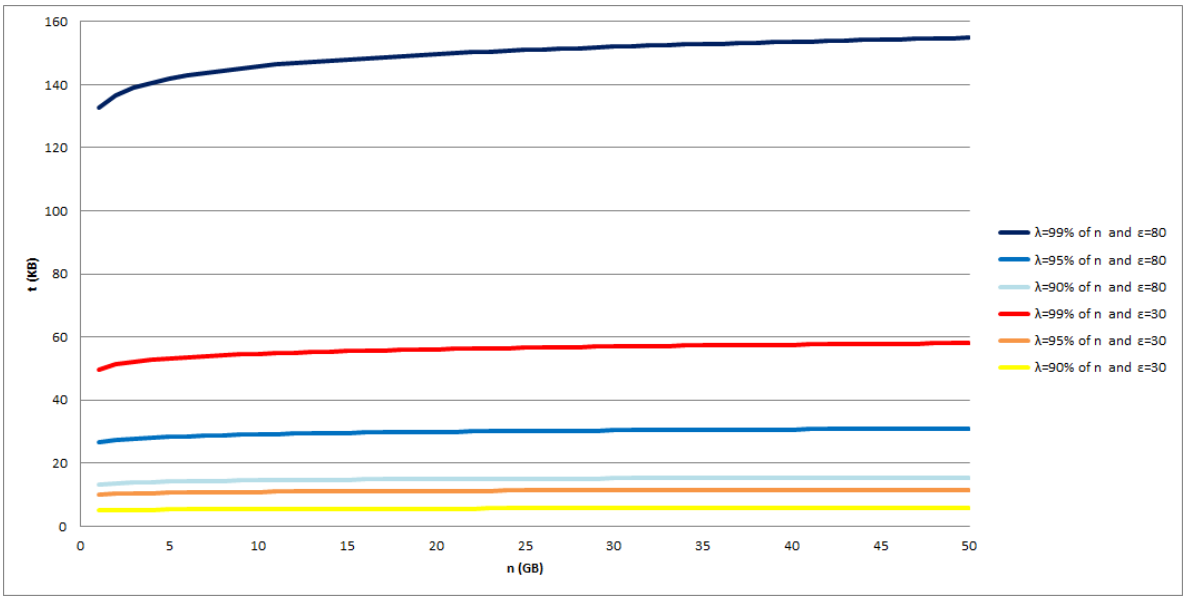
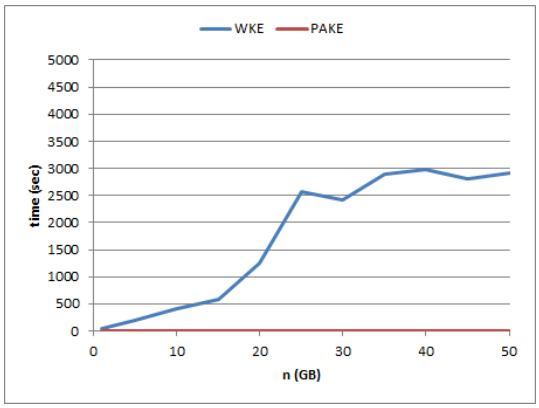


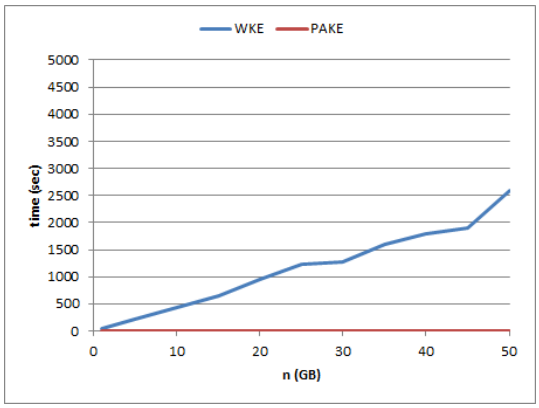

**Fig. 4.** Comparison between the values of $t$ in the standard setting $\epsilon = 30$ and in an higher level of security setting $\epsilon = 80$, for file size $1GB \leq n \leq 50GB$ and leakage $90\%$ of $n \leq \lambda \leq 99\%$ of $n$.

In Figure 5, we show the experimental evaluation of the running time of the AKE protocol, comparing the standard setting $\epsilon = 30$ with an higher level of security setting $\epsilon = 80$, for the same values of $n$ and $\lambda$. From this comparison, we note that the request of an higher level of security does not affect significantly the efficiency of our AKE protocol. In particular, we note that unexpectedly, for some values of $n$, the running time of the protocol for the setting $\epsilon = 80$ is smaller than the case in which we set $\epsilon = 30$, i.e. the protocol is faster. This can be explained as if the value of $t$ increases, even if it is still very small compared to the size of the shared file, then the bits that the parties have to extract from the file become less sparse and therefore the extraction of the bits from the file can be optimized by the operative system. Indeed, in one block of the file there are, with higher probability, more bits to extract and therefore there is not needed to access the disk to extract them, as they are already in memory, buffered by the operative system. Clearly, reading bits from the memory is faster than access the disk. Further, we would like to note that, despite the fact that the total amount of time needed to run the protocol looks expensive (more than 40 minutes for large $n$), a very large amount of time is used to perform the WKE protocol, while the PAKE protocol is drastically faster (it takes only from few seconds to less than a minute). The WKE protocol is expensive, in terms of time, because it performs the accesses to the disk, the most expensive operation of our protocol. Therefore, to improve the efficiency of the AKE protocol it is sufficient to optimize this operation, e.g. using hardware which allows faster disk access time or mapping

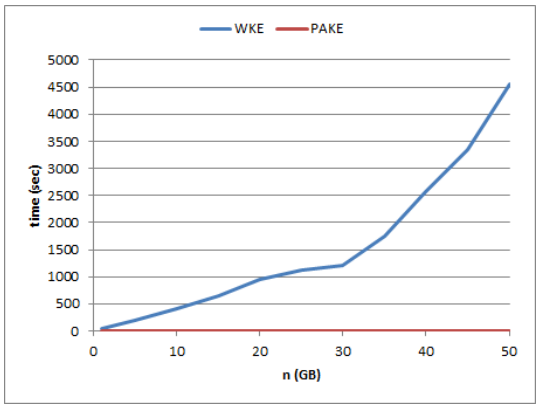
[3] Client and Server access (at most) $2t$ bits in the file: $t$ indexes chosen by Client and $t$ indexes chosen by Server. Some indexes (even all of them) might be equal.



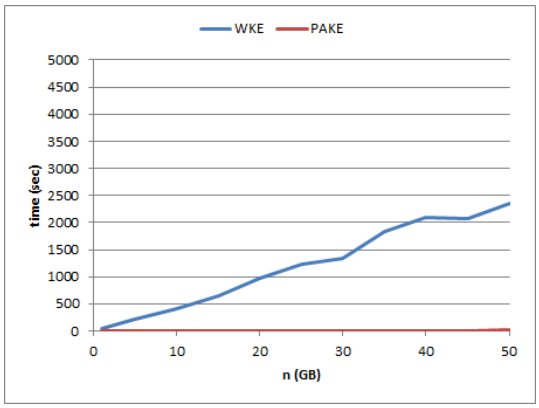

(a) $\epsilon = 30$ and $\lambda = 90\%$ of $n$



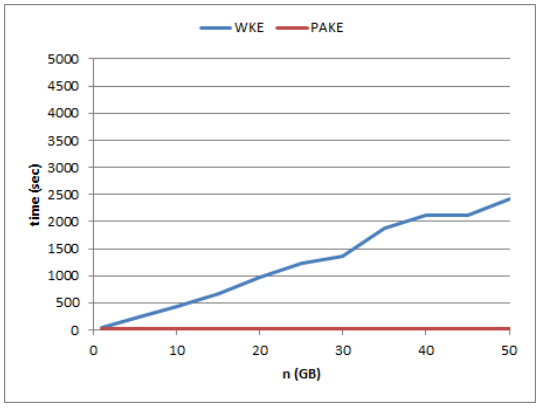

(b) $\epsilon = 80$ and $\lambda = 90\%$ of $n$



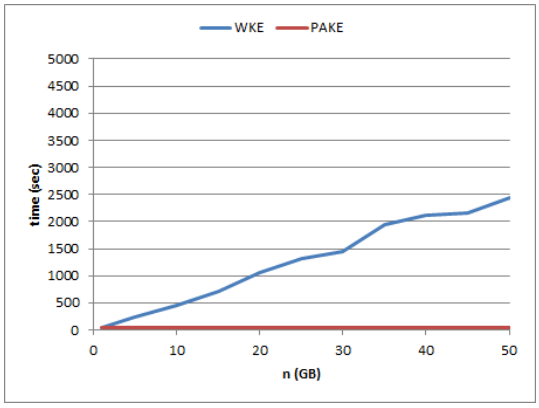

(c) $\epsilon = 30$ and $\lambda = 95\%$ of $n$




(d) $\epsilon = 80$ and $\lambda = 95\%$ of $n$




(e) $\epsilon = 30$ and $\lambda = 99\%$ of $n$




(f) $\epsilon = 80$ and $\lambda = 99\%$ of $n$

**Fig. 5.** Experimental evaluation of the running time of the WKE and PAKE protocols. Comparison between the standard setting $\epsilon = 30$ and an higher level of security setting $\epsilon = 80$, for file size $1GB \leq n \leq 50GB$. Figure 5(a) and 5(b) considers $\lambda = 90\%$ of $n$, Figure 5(c) and 5(d) considers $\lambda = 95\%$ of $n$ and Figure 5(e) and 5(f) considers $\lambda = 99\%$ of $n$.

the whole shared file in the memory (but then one should study also the additional leakage of such operation). However, in section 4.3 we show that Client and Server needs to compute the shared keys only once, as the keys remain secure even if, after the end of the AKE protocol, the adversary retrieves all the shared file.

### 4.2 Password-based Authenticated Key Exchange

In our implementation of the AKE protocol, we use the UC-PAKE protocol of Abdalla et al. showed in [1]. As noted in [25], we need to use a UC-PAKE protocol instead of simply a standard PAKE protocol as the passwords output by the WKE protocol and then used by the PAKE protocol, in our implementation of the AKE protocol, could be arbitrary correlated and, unlike more traditional models, in the UC framework there are "no assumptions on the distribution on the passwords" used by the parties. In [1], the authors presented an "ideal functionality" $\mathcal{F}_{PAKE}^{CAuth}$ (see Figure 6) for PAKE protocol with client authentication (which extends that one in [23]) and then they proved that the protocol showed in Section 3.2 securely realizes such "ideal functionality"[4], against adaptive adversaries. To provide a random oracle and an ideal cipher to the parties, in their proof Abdalla et al. use two ideal functionalities $\mathcal{F}_{RO}$ (already introduced in [40]) and $\mathcal{F}_{IC}$, such that the random oracle model and the ideal model UC-emulates $\mathcal{F}_{RO}$ and $\mathcal{F}_{IC}$ respectively. The functionality $\mathcal{F}_{PAKE}^{CAuth}$ provides that:

- the parties receive the same uniformly distributed random key if they share the same password and the adversary does not corrupt one of them and she does not make any attempts to guess their passwords;
- Server receives an error message if the parties share different passwords or the adversary tries to guess the parties' passwords but fails (she has only one attempt) [5];
- the parties receive the same key chosen by the adversary if the adversary corrupts one party (and the parties share the same password) or she correctly guesses a party's password (with only one try allowed).

Moreover, $\mathcal{F}_{PAKE}^{CAuth}$ allows an adversary, who does not corrupt the players and who does not correctly guess their passwords, to learn only when the parties start the PAKE protocol. We note that even if the adversary simply fails to guess the password of one party, without compromises the protocol, then the Server aborts the protocol. In this way, the functionality provides the client-to-server authentication, as Server accepts a key only if Client shares the same one.

As the functionality of [1] does not provide the passwords to the parties, it models the case in which the parties run the protocol with different or correlated passwords.

### 4.3 Authenticated Key Exchange

In this section, following the proof sketched in [25], we show that the keys output by the AKE protocol of Figure 3 are indistinguishable from random to an adversary point of view (and equal), if the parties accept them.

In our AKE protocol, we use a UC-PAKE protocol, therefore we can substitute its execution with calls to the ideal functionality $\mathcal{F}_{PAKE}^{CAuth}$ of Figure 6, using $Pwd_C$ and $Pwd_S$ of Figure 1, the weak keys output by the WKE protocol, as the passwords for the `NewSession` query of $P_i$ and $P_j$, respectively. By the definition of $\mathcal{F}_{PAKE}^{CAuth}$, only if an adversary corrupts a party or correctly guesses the password of one of the parties (before the end of the PAKE protocol) then she can choose the output keys of the parties. Otherwise, the keys output by the ideal functionality are indistinguishable from random to the adversary (or Server simply does not accept any key). To guess the passwords used by the parties in the PAKE protocol, the adversary has only one attempt, using the `TestPwd` query (cf. Figure 6). By the analysis of the WKE protocol (see Lemma 4), we know that

[4] Technically, they prove that the protocol securely realizes the multi-session extension of the functionality, in the joint state version of the UC framework.

[5] However, Client receives a uniformly distributed random key as the authentication is only client-to-server.

$\mathcal{F}_{PAKE}^{CAuth}$ owns a list $L$ initially empty of values of the form $(P_i, P_j, pwd)$.

- **Upon receiving a query (`NewSession`, $ssid$, $P_i$, $P_j$, $pwd$, `role`) from $P_i$:**
  - Send (`NewSession`, $ssid$, $P_i$, $P_j$, `role`) to $\mathcal{A}$.
  - If this is the first `NewSession` query, or if it is the second `NewSession` query and there is a record $(P_j, P_i, pwd', \overline{\texttt{role}}) \in L$, then record $(P_i, P_j, pwd, \texttt{role})$ in $L$ and mark this record `fresh`.
- **Upon receiving a query (`TestPwd`, $ssid$, $P_i$, $pwd'$) from the adversary $\mathcal{A}$:**
  If there exists a record of the form $(P_i, P_j, pwd, \texttt{role}) \in L$ which is `fresh`, then do:
  - If $pwd = pwd'$, mark the record `compromised` and reply to $\mathcal{A}$ with "correct guess".
  - If $pwd \neq pwd'$, mark the record `interrupted` and reply to $\mathcal{A}$ with "wrong guess".
- **Upon receiving a query (`NewKey`, $ssid$, $P_i$, $sk$) from $\mathcal{A}$, where $|sk| = k$:**
  If there is a record of the form $(P_i, P_j, pwd, \texttt{role}) \in L$, and this is the first `NewKey` query for $P_i$, then:
  If $\texttt{role} = \texttt{client}$ :
  - If the session is `compromised`, or if one of the two players $P_i$ or $P_j$ is corrupted, then send $(ssid, sk)$ to $P_i$, record $(P_i, P_j, pwd, \texttt{client}, \texttt{completed})$ in $L$, as well as $(ssid, P_i, pwd, sk, \texttt{client}, \texttt{status}, \texttt{ready})$ (with `status` being the status of the session at that moment.
  - Else, if the session is `fresh` or `interrupted`, choose a random key $sk'$ whose length is $k$ and send $(ssid, sk')$ to $P_i$. Record $(P_i, P_j, pwd, \texttt{client}, \texttt{completed})$ in $L$, as well as $(ssid, P_i, pwd, sk', \texttt{client}, \texttt{status}, \texttt{ready})$ where `status` stands for `fresh` or `interrupted`;

  If $\texttt{role} = \texttt{server}$ :
  - If the session is `compromised`, if one of the two players $P_i$ or $P_j$ is corrupted, and if there are two records of the form $(P_i, P_j, pwd, \texttt{server})$ and $(P_j, P_i, pwd', \texttt{client})$, set $s = sk$. Otherwise, if the session is `fresh` and there exists any recorded element of the form $(ssid, P_j, pwd', sk', \texttt{client}, \texttt{fresh}, \texttt{ready})$, set $s = sk'$.
    - If $pwd = pwd'$, send $(ssid, s)$ to $P_i$, record $(P_i, P_j, pwd, \texttt{server}, \texttt{completed})$ in $L$, as well as $(ssid, P_i, pwd, s, \texttt{server}, \texttt{status})$.
    - If $pwd \neq pwd'$, send $(ssid, \texttt{error})$ to $P_i$, record $(P_i, P_j, pwd, \texttt{server}, \texttt{completed})$ in $L$, as well as $(ssid, P_i, pwd, \texttt{server}, \texttt{status})$.
  - if the session is `fresh` and there does not exist any recorded element of the form $(ssid, P_j, pwd', sk', \texttt{client}, \texttt{fresh}, \texttt{ready})$, then do not do anything.
  - If the session is `interrupted` then send $(ssid, \texttt{error})$ to player $P_i$, and record in $L$ $(P_i, P_j, pwd, \texttt{server}, \texttt{completed})$ and $(ssid, P_i, pwd, \texttt{server}, \texttt{error}, \texttt{completed})$.

**Fig. 6.** Ideal functionality $\mathcal{F}_{PAKE}^{CAuth}$: it is parametrizes by a security parameter $k$. It interacts with an adversary $\mathcal{A}$ and a set of parties $P_1, \ldots, P_n$. (from [1])

the passwords output by the WKE protocol, and then used in the PAKE protocol, have (with high probability $1 - 2^{-\epsilon}$, where $\epsilon$ is the security parameter) a min-entropy equals to $\epsilon$ conditioned on the adversary view before the call to $\mathcal{F}_{PAKE}^{CAuth}$, which is equal to the adversarial view after the WKE protocol plus the nonces exchanged between the parties to create the random $ssid$, and this holds even after that the adversary retrieves up to $\lambda$ bits from the internal state of the parties and the shared file $F$, before the beginning of the WKE protocol. Therefore, an adversary can correctly guess in one attempt the passwords used in the PAKE protocol only with very low probability, chosen by Client. Then we can conclude that the keys $sk_C$ and $sk_S$ obtained by the parties at the end of the AKE protocol, are indistinguishable from random to the adversary. Moreover, these keys are equal as otherwise Server receives an error message (and rejects the key), providing the client-to-server authentication.

Finally, we note that even after the end of the AKE protocol, the adversary can never distinguish $sk_C$ and $sk_S$ from random. Indeed, if, after the end of the AKE protocol, an adversary retrieves the bits of the shared file used in the AKE protocol to compute the passwords output by the WKE protocol (without violating the retrieval bound), she can clearly compute the passwords but still she cannot distinguish the secret keys obtained by the parties from random, as they have been chosen at random by the functionality $\mathcal{F}_{PAKE}^{CAuth}$. If we look at the UC-PAKE protocol, it means that even if the adversary learns, after the execution of the protocol, the value of the passwords used in the protocol, she cannot distinguish the output keys from random, as the protocol, from her point of view, becomes a DH Key Exchange protocol, as now she can decrypt the Server's public DH value, and we rely on the CDH assumption.Moreover, even if the adversary is given all the shared file, still the security guarantees of the completed AKE protocol hold but then, of course, there is no more security for future running of the protocol, as the retrieval bound requirement is gone.

## References


1. Michel Abdalla, Dario Catalano, Céline Chevalier, and David Pointcheval. Efficient two-party password-based key exchange protocols in the UC framework. In Tal Malkin, editor, *CT-RSA*, volume 4964 of *Lecture Notes in Computer Science*, pages 335–351. Springer, 2008.
2. Michel Abdalla and David Pointcheval. Simple password-based encrypted key exchange protocols. In Alfred Menezes, editor, *CT-RSA*, volume 3376 of *Lecture Notes in Computer Science*, pages 191–208. Springer, 2005.
3. Joël Alwen, Yevgeniy Dodis, Moni Naor, Gil Segev, Shabsi Walfish, and Daniel Wichs. Public-key encryption in the bounded-retrieval model. In Henri Gilbert, editor, *EUROCRYPT*, volume 6110 of *Lecture Notes in Computer Science*, pages 113–134. Springer, 2010.
4. Joël Alwen, Yevgeniy Dodis, and Daniel Wichs. Leakage-resilient public-key cryptography in the bounded-retrieval model. In Shai Halevi, editor, *CRYPTO*, volume 5677 of *Lecture Notes in Computer Science*, pages 36–54. Springer, 2009.
5. Mihir Bellare and Sara K. Miner. A forward-secure digital signature scheme. In Michael J. Wiener, editor, *CRYPTO*, volume 1666 of *Lecture Notes in Computer Science*, pages 431–448. Springer, 1999.
6. Mihir Bellare, David Pointcheval, and Phillip Rogaway. Authenticated key exchange secure against dictionary attacks. In Preneel [51], pages 139–155.
7. Mihir Bellare and Phillip Rogaway. Entity authentication and key distribution. In Douglas R. Stinson, editor, *CRYPTO*, volume 773 of *Lecture Notes in Computer Science*, pages 232–249. Springer, 1993.
8. Mihir Bellare and Phillip Rogaway. Provably secure session key distribution: the three party case. In *STOC '95: Proceedings of the Twenty-Seventh Annual ACM Symposium on Theory of Computing, 29 May-1 June 1995, Las Vegas, Nevada, USA*, pages 57–66. ACM, 1995.
9. Mihir Bellare and Phillip Rogaway. The AuthA protocol for password-based authenticated key exchange. In *Contributions to IEEE P1363*, 2000.
10. Mihir Bellare and John Rompel. Randomness-efficient oblivious sampling. In *FOCS '94: Proceedings of 35th Annual Symposium on Foundations of Computer Science, 20-22 November 1994, Santa Fe, New Mexico, USA*, pages 276–287. IEEE, 1994.

11. Mihir Bellare and Bennet S. Yee. Forward-security in private-key cryptography. In Marc Joye, editor, *CT-RSA*, volume 2612 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2003.
12. Steven M. Bellovin and Michael Merritt. Augmented encrypted key exchange: A password-based protocol secure against dictionary attacks and password file compromise. In *ACM Conference on Computer and Communications Security*, pages 244–250, 1993.
13. Dan Boneh, editor. *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings*, volume 2729 of *Lecture Notes in Computer Science*. Springer, 2003.
14. Xavier Boyen, Yevgeniy Dodis, Jonathan Katz, Rafail Ostrovsky, and Adam Smith. Secure remote authentication using biometric data. In Cramer [26], pages 147–163.
15. Victor Boyko, Philip D. MacKenzie, and Sarvar Patel. Provably secure password-authenticated key exchange using diffie-hellman. In Preneel [51], pages 156–171.
16. Emmanuel Bresson, Olivier Chevassut, and David Pointcheval. Security proofs for an efficient password-based key exchange. In Sushil Jajodia, Vijayalakshmi Atluri, and Trent Jaeger, editors, *ACM Conference on Computer and Communications Security*, pages 241–250. ACM, 2003.
17. Emmanuel Bresson, Olivier Chevassut, and David Pointcheval. New security results on encrypted key exchange. In Feng Bao, Robert H. Deng, and Jianying Zhou, editors, *Public Key Cryptography*, volume 2947 of *Lecture Notes in Computer Science*, pages 145–158. Springer, 2004.
18. A. Brusilovsky, I. Faynberg, and Z. Zeltsan. Password-Authenticated Key (PAK) Diffie-Hellman Exchange RFC5683, February 2010.
19. Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS*, pages 136–145, 2001.
20. Ran Canetti, Yevgeniy Dodis, Shai Halevi, Eyal Kushilevitz, and Amit Sahai. Exposure-Resilient Functions and All-Or-Nothing Transforms. In *EUROCRYPT*, pages 453–469, 2000.
21. Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited. *J. ACM*, 51(4):557–594, 2004.
22. Ran Canetti, Shai Halevi, and Jonathan Katz. A forward-secure public-key encryption scheme. *J. Cryptology*, 20(3):265–294, 2007.
23. Ran Canetti, Shai Halevi, Jonathan Katz, Yehuda Lindell, and Philip D. MacKenzie. Universally Composable password-based key exchange. In Cramer [26], pages 404–421.
24. Ran Canetti and Tal Rabin. Universal composition with joint state. In Boneh [13], pages 265–281.
25. David Cash, Yan Zong Ding, Yevgeniy Dodis, Wenke Lee, Richard J. Lipton, and Shabsi Walfish. Intrusion-resilient key exchange in the bounded retrieval model. In Salil P. Vadhan, editor, *TCC*, volume 4392 of *Lecture Notes in Computer Science*, pages 479–498. Springer, 2007.
26. Ronald Cramer, editor. *Advances in Cryptology - EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005, Proceedings*, volume 3494 of *Lecture Notes in Computer Science*. Springer, 2005.
27. Giovanni Di Crescenzo, Richard J. Lipton, and Shabsi Walfish. Perfectly secure password protocols in the bounded retrieval model. In Halevi and Rabin [39], pages 225–244.
28. Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, November 1976.
29. Yevgeniy Dodis, Matthew K. Franklin, Jonathan Katz, Atsuko Miyaji, and Moti Yung. Intrusion-resilient public-key encryption. In *CT-RSA*, pages 19–32, 2003.
30. Yevgeniy Dodis, Jonathan Katz, Leonid Reyzin, and Adam Smith. Robust fuzzy extractors and authenticated key agreement from close secrets. In Cynthia Dwork, editor, *CRYPTO*, volume 4117 of *Lecture Notes in Computer Science*, pages 232–250. Springer, 2006.
31. Yevgeniy Dodis, Jonathan Katz, Shouhuai Xu, and Moti Yung. Key-insulated public key cryptosystems. In Lars R. Knudsen, editor, *EUROCRYPT*, volume 2332 of *Lecture Notes in Computer Science*, pages 65–82. Springer, 2002.
32. Yevgeniy Dodis, Jonathan Katz, Shouhuai Xu, and Moti Yung. Strong key-insulated signature schemes. In Yvo Desmedt, editor, *Public Key Cryptography*, volume 2567 of *Lecture Notes in Computer Science*, pages 130–144. Springer, 2003.

33. Yevgeniy Dodis, Rafail Ostrovsky, Leonid Reyzin, and Adam Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. *SIAM J. Comput.*, 38(1):97–139, 2008.
34. Yevgeniy Dodis, Amit Sahai, and Adam Smith. On perfect and adaptive security in exposure-resilient cryptography. In Pfitzmann [50], pages 301–324.
35. Stefan Dziembowski. Intrusion-resilience via the bounded-storage model. In Halevi and Rabin [39], pages 207–224.
36. Stefan Dziembowski and Krzysztof Pietrzak. Intrusion-resilient secret sharing. In *FOCS '07: Proceedings of 48th Annual IEEE Symposium on Foundations of Computer Science, October 20-23, 2007, Providence, RI, USA*, pages 227–237. IEEE Computer Society, 2007.
37. Rosario Gennaro and Yehuda Lindell. A framework for password-based authenticated key exchange. In Eli Biham, editor, *EUROCRYPT*, volume 2656 of *Lecture Notes in Computer Science*, pages 524–543. Springer, 2003.
38. Oded Goldreich and Yehuda Lindell. Session-key generation using human passwords only. In Joe Kilian, editor, *CRYPTO*, volume 2139 of *Lecture Notes in Computer Science*, pages 408–432. Springer, 2001.
39. Shai Halevi and Tal Rabin, editors. *Theory of Cryptography, Third Theory of Cryptography Conference, TCC 2006, New York, NY, USA, March 4-7, 2006, Proceedings*, volume 3876 of *Lecture Notes in Computer Science*. Springer, 2006.
40. Dennis Hofheinz and Jörn Müller-Quade. Universally composable commitments using random oracles. In Moni Naor, editor, *TCC*, volume 2951 of *Lecture Notes in Computer Science*, pages 58–76. Springer, 2004.
41. IEEE Standard 1363.2 Study Group. Password-based public-key cryptography. Webpage: `http://grouper.ieee.org/groups/1363/passwdPK/` accessed on 5.1.2012.
42. Gene Itkis and Leonid Reyzin. Sibir: Signer-base intrusion-resilient signatures. In Moti Yung, editor, *CRYPTO*, volume 2442 of *Lecture Notes in Computer Science*, pages 499–514. Springer, 2002.
43. B. Kaliski. PKCS #5: Password-Based Cryptography Specification. Version 2.0. RFC2898, September 2000.
44. Jesse Kamp and David Zuckerman. Deterministic extractors for bit-fixing sources and exposure-resilient cryptography. *SIAM J. Comput.*, 36(5):1231–1247, 2007.
45. Jonathan Katz, Rafail Ostrovsky, and Moti Yung. Efficient password-authenticated key exchange using human-memorable passwords. In Pfitzmann [50], pages 475–494.
46. Jonathan Katz, Rafail Ostrovsky, and Moti Yung. Forward secrecy in password-only key exchange protocols. In Stelvio Cimato, Clemente Galdi, and Giuseppe Persiano, editors, *SCN*, volume 2576 of *Lecture Notes in Computer Science*, pages 29–44. Springer, 2002.
47. Philip MacKenzie. The pak suite: Protocols for password-authenticated key exchange. In *IEEE P1363.2*, 2002.
48. Ueli M. Maurer and Stefan Wolf. Secret-key agreement over unauthenticated public channels III: Privacy amplification. *IEEE Transactions on Information Theory*, 49(4):839–851, 2003.
49. OpenSSL project webpage:. `http://www.openssl.org` accessed on 5.1.2012.
50. Birgit Pfitzmann, editor. *Advances in Cryptology - EUROCRYPT 2001, International Conference on the Theory and Application of Cryptographic Techniques, Innsbruck, Austria, May 6-10, 2001, Proceeding*, volume 2045 of *Lecture Notes in Computer Science*. Springer, 2001.
51. Bart Preneel, editor. *Advances in Cryptology - EUROCRYPT 2000, International Conference on the Theory and Application of Cryptographic Techniques, Bruges, Belgium, May 14-18, 2000, Proceeding*, volume 1807 of *Lecture Notes in Computer Science*. Springer, 2000.
52. Renato Renner and Stefan Wolf. Unconditional authenticity and privacy from an arbitrarily weak secret. In Boneh [13], pages 78–95.
53. V. Shoup. On formal models for secure key exchange (Version 4). Technical report RZ 3120. IBM Research Zurich, 1999.
54. Salil P. Vadhan. Constructing locally computable extractors and cryptosystems in the bounded-storage model. *J. Cryptology*, 17(1):43–77, 2004.
55. Stefan Wolf. Strong security against active attacks in information-theoretic secret-key agreement. In Kazuo Ohta and Dingyi Pei, editors, *ASIACRYPT*, volume 1514 of *Lecture Notes in Computer Science*, pages 405–419. Springer, 1998.