PUF-based mutual multifactor entity and transaction authentication for secure banking

Amanda C. Davi Resende, Karina Mochetti, and Diego F. Aranha*

Institute of Computing – University of Campinas (UNICAMP)
amanda@lasca.ic.unicamp.br, {mochetti,dfaranha}@ic.unicamp.br

Abstract. In this work propose a protocol combining a Physical Unclonable Function (PUF) with Password-based Authenticated Key Exchange (PAKE). The resulting protocol provides mutual multifactor authentication between client and server and establishes a session key between the authenticated parties, important features that were not found simultaneously in the literature of PUF-based authentication. The combination can be adapted to support a panic password which allows the client to notify the server in case of emergency. Moreover, a novel protocol for two-factor transaction authentication is proposed. This ensures that only parties authenticated in the current session can realize valid bank transactions.

Keywords: PUF, PAKE, multifactor authentication, secure banking.

1 Introduction

One of the main concerns in modern cryptography is that one or both communicating parties can prove its identity to the other party. *Authentication* protocols can be used for this purpose and generally depend on the knowledge of a long cryptographic key of exclusive possession by the legitimate holders. For this reason, it is common to employ a device to store this key securely, since human memory, in most cases, is unable to memorize it without errors. Protocols for Password-based Authenticated Key Exchange (PAKE) relax this requirement to the extent that they require the knowledge of a much shorter key (*password*).

Authentication protocols can be applied to several scenarios, ranging from simply obtaining access to a computer, to securing a bank transaction. The applicability of these protocols in the banking environment is extremely important, since many features are provided to the client via the Internet and ATMs, in which the bank primarily needs to confirm the authenticity of the client's identity. Conversely, the client must verify the authenticity of the bank's identity before providing credentials or sharing financial information. In practice, most banking security solutions for client authentication involve an token producing

* The authors thank Intel Labs for funding the project "*Physical Unclonable Functions for SoC Devices*" in which scope this work was conducted.

one-time passwords generated by a synchronized timer and/or some keyed function. Server authentication is performed in parallel, through certificates in the SSL/TLS protocol. The main limitations of these client-side solutions are vulnerability against reverse engineering for key extraction, lack of challenge-response mechanisms, and difficult integration with other authentication factors. In this work, we argue that exploring unpredictable physical effects improves security by removing the need of explicitly stored secrets that may leak or be captured by an adversary.

Several works in the literature employ Physical Unclonable Functions (PUF) [1,2] for constructing leakage-resilient block ciphers [3], performing device authentication [4], generating cryptographic keys [5], among others. Some works aim to replace the possession of a long-term cryptographic key with a PUF in authentication protocols. Delvaux *et. al.* [6] surveys lightweight authentication protocols and presents desirable requirements and attacks against many protocols in this solution space. Tuyls and Škorić [7] proposed a PUF-based protocol for authentication in banking applications with establishment of a session key. However, this protocol is not secure against server impersonation attacks in a realistic adversarial model, in which the adversary has physical access to the PUF, as shown by Busch *et al.* [8]. In Section 3, we show that the correction proposed by Busch *et al.* [8] for the Tuyls and Škorić protocol [7] still retains the vulnerability against server impersonation. Another work by Frikken *et al.* [9] employs a zero-knowledge proof of possession of a PUF for client authentication through a bank-issued device and an additional password. Beyond not establishing a session key and offering only client authentication, instead of mutual authentication, this protocol also vulnerable to offline dictionary attacks, as discussed in Section 3. As a result, these protocols cannot be considered secure for a realistic attacker, who has temporary possession of the PUF and colludes with an attacker able to monitor network traffic between client and server.

This paper proposes PUF+PAKE, a secure protocol resulting from the combination between PUF and PAKE. This combination is sound, because the PAKE only requires knowledge of a small shared password, and the PUF output may not be large enough to be comparable to a cryptographic key. The general construction uses the PUF output as the shared password required by the PAKE, ensuring that the shared session key produced by the PAKE will only be available under possession of the PUF, and improving leakage resistance by eliminating any long-term secrets stored explicitly. The protocol will also be protected against dictionary offline attacks by employing the PAKE. In particular, the session key can be used to protect subsequent communications involving financial information of the client. In particular, our protocols provably satisfy the following security requirements: (i) no long-term secret needs to be stored; (ii) a user should be unable to successfully authenticate without his/her device; (iii) a stolen device cannot be used to impersonate the user; (iv) and the protocol must have protection of additional credentials against offline attacks, a property hard to obtain with a lightweight protocol. The contributions of this paper are:

- A server impersonation attack against the Busch *et al.* protocol [8] which shows that the correction suggested by the authors for Tuyls and Škorić protocol [7] is not sufficient. This is a new attack not presented by Delvaux *et al.* [6].
- Dictionary attacks against the protocol proposed by Frikken *et al.* [9] that depend only on the temporary possession of the PUF and observation of a single trace of client-server communication.
- A protocol combining PUF and PAKE for mutual multifactor[1] (using the output of the PUF and the user's password) authentication and session key establishment between authenticated parties. The protocol combines several important features present in other protocols but that were not found together in previous protocols and offers an interesting security trade-off when compared to related work: lower resistance against internal agents, but enhanced security against dictionary attacks. The proposed solution is particularly applicable to the banking sector, where clients are already used to have an additional authentication device. Current devices could then be augmented with a PUF to provide multifactor authentication based on both computational and physical assumptions.
- Formal security analysis of the protocol, considering standard security notions for authenticated key exchange.
- An adaptation of the proposed protocol to allow the client to notify the server in case of emergency.
- A novel protocol for two-factor authentication of bank transactions, requiring knowledge of the session key and an updated proof of possession of the PUF as authentication factors.

This paper is organized as follows. In Section 2, we present the definitions used during the development of this work. In Section 3, we show attacks against two known protocols. The proposed protocol and its formal security analysis are described in Sections 4 and 5, respectively. Section 6 presents a simple adaptation of the protocol to support panic passwords. Section 7 presents our PUF-based solution for transaction authentication and its security analysis. Section 8 presents comparison and improvements over the protocols attacked on Section 3. Finally, in Section 9, we present the conclusions and point out directions for future work.

2 Preliminary definitions

2.1 Physical Unclonable Functions

A Physical Unclonable Function [1,2] is usually implemented through an unpredictable physical effect intrinsically linked to each individual instantiation. By assumption, PUFs cannot be easily duplicated or manipulated without changing their behavior considerably. Despite the existence of PUFs with certain security

[1] We consider an authentication *factor* as any additional credential (such as biometric information).

properties being contested in the literature [10,11,12], our work depends on a suitable choice of a PUF that satisfies rigorous security properties. Although PUF behavior may be influenced by environmental factors, such as changes in the temperature, we argue that *there is a good PUF candidate* [13] that can be made reliable enough for actual deployment. In this work, we do not specify a PUF to not lose generality and for simplifying the security analysis. Notice that a strong PUF natively supporting a physically complex challenge-response mechanism is required.

Since these physical effects are rarely completely stable, a PUF can generate slightly different outputs for queries with the same input at two distinct points of time. This noisy aspect hinders the direct use of PUFs in cryptographic applications, but fuzzy extractors [14] enable the conversion of PUF outputs to a close-to-uniform distribution of binary strings. A fuzzy extractor is a pair of probabilistic procedures to generate $Gen : \{0,1\}^n \rightarrow \{0,1\}^\ell \times \{0,1\}^*$ and reproduce $Rep : \{0,1\}^n \times \{0,1\}^* \rightarrow \{0,1\}^\ell$ PUF outputs, where the generation process produces auxiliary information ω for which its output can be recovered with the Rep procedure. More formally, we fix a (m, ℓ, t, ϵ) fuzzy extractor equipped with these procedures that are able to receive any distribution of inputs d with min-entropy m and generate $(r, \omega) \leftarrow Gen(d)$ with statistical difference between (r, ω) and $(\mathbb{U}_\ell, \omega)$ at most negligible ϵ. The correctness property requires the Rep procedure to exactly reproduce $r \leftarrow Rep(d', \omega)$ when $dist(d, d') \leq t$.

Given its unpredictable behavior, PUFs can be mathematically modeled as a function $PUF : \{0,1\}^m \rightarrow \{0,1\}^n$, for which one can define the following response game against a polynomial adversary $\mathcal{A}$ [9]:

- **Phase 1**: The adversary $\mathcal{A}$ requests and receives responses (r_i, ω_i) for any PUF challenge d_i of its choice.
- **PUF challenge**: $\mathcal{A}$ chooses a PUF challenge d not queried previously and receives auxiliary information ω produced by $Gen(PUF(d))$, but not its output.
- **Phase 2**: $\mathcal{A}$ can do more requests for the PUF for any other PUF challenges different from d.
- **Responde**: Eventually, $\mathcal{A}$ outputs its guess r' for $r = Rep(PUF(d), \omega)$.

The adversary $\mathcal{A}$ wins if $r = r'$. For an unpredictable PUF, $\mathcal{A}$ has the winning probability $Adv_{\mathcal{A}}^{PUF}(\ell) = Pr[r = r']$ as a negligible function of the security parameter ℓ. For the *PUF response game* above, a decisional version can be defined as the *PUF response indistinguishability game* as follows [9]:

- **Enroll**: $\mathcal{A}$ executes the enrollment phase for any value d_i of its choice, receiving the corresponding ω_i. Define the set of such pairs (d_i, ω_i) as $\mathcal{W}$.
- **Phase 1**: $\mathcal{A}$ requests and receives PUF responses r_i for any $(d_i, \omega_i) \in \mathcal{W}$ of its choice.
- **PUF challenge**: $\mathcal{A}$ chooses a PUF challenge d that has been queried during the **Enroll** phase but not in **Phase 1**. A random bit b is chosen. If $b = 0$, $\mathcal{A}$ receives $r = Rep(PUF(d), \omega)$, where $(d, \omega) \in \mathcal{W}$, otherwise it receives a string uniformly chosen from $\{0,1\}^\ell$.

- **Phase 2** $\mathcal{A}$ is allowed to query the PUF for challenges in $\mathcal{W}$ other than (d, ω).
- **Response**: Eventually, $\mathcal{A}$ outputs a bit b'.

The adversary $\mathcal{A}$ wins if $b = b'$. Let $Adv_{\mathcal{A}}^{PUF-ind} = Pr[b = b']$ be the probability of $\mathcal{A}$ winning the game. It is assumed that $Adv_{\mathcal{A}}^{PUF-ind}(\ell) - \frac{1}{2}$ is also negligible.

2.2 Password-based Authenticated Key Exchange

A Password-based Authenticated Key Exchange protocol establishes a shared key over an insecure channel depending only on a small shared secret, called a *password*. This is a very useful feature because an extra device is not required for storing a long cryptographic key, but only the ability of human memory to store a short secret.

The first PAKEs arised in the 90s, with the Diffie-Hellman Encrypted Key Exchange (DH-EKE) protocol in 1992 [15] and the Simple Password Exponential Key Exchange (SPEKE) protocol in 1996 [16]. More recently, an important PAKE was proposed as the AuthA protocol [17] that provides the same security properties of previous protocols, but is simpler, has a lower cost of communication and is more versatile. Other PAKE protocols can be found in the literature [18,19], with some based on the AuthA protocol [20,21]. In this paper, we employ a simplified variant of a AuthA protocol [20,22] as choice of PAKE for the purpose of illustration, but any secure PAKE protocol can be used instead.

Semantic security. Protocols are subject to various attacks, because the message exchanges occur over an insecure channel. The information transmitted is subject to various types of threats such as eavesdropping and tampering. A malicious agent can eavesdrop honest conversation between two entities and record all exchanges of messages for later impersonating one of the two parties. This type of attack is called a *replay* or *repetition attack*.

One way to avoid this type of attack is to ensure that the protocols provide the property of *freshness*. This property ensures that the messages probably belong to the current execution, and do not constitute repetitions of previous messages exchanged in some other honest execution. The freshness property can be obtained in various ways, such as the use of clocks like in the Kerberos authentication system [23,24] or the use of challenges-response operations [25].

Challenge-response mechanisms are commonly used because they do not require clock synchronization between the entities making them robust and popular in cryptographic protocol design. In this method, an entity A sends a random value (challenge) to an entity B and requires this value to be in the next message (response) received from B, and contrariwise. The challenge must be protected so that it can only be read by the legitimate recipient. This can be done in several ways [16,18,20]. For example, the client can choose a random value $x \in [1, q-1]$ where q is the size of a finite cyclic group $\mathbb{G}$, compute the value of g^x for a generator $g \in \mathbb{G}$ and send it to the server, which in turn, chooses a random value

$y \in [1, q-1]$, computes g^y, encrypts the result using a shared secret and sends the result to the client. Afterwards, each party verifies implicitly if the other party received the correct nonce.

Authentication. One of the goals of authenticated key exchange protocols is to ensure the authentication property, which guarantees that the shared cryptographic key is obtained only by parties that satisfy the authentication requirements. The probability of an adversary $\mathcal{A}$ to impersonate the client or the server in a protocol run P is denoted by $Adv_{\mathcal{A},P}^{m-auth}(\ell)$ and thus, the protocol P is said to be *secure* if this probability is negligible in the security parameter ℓ.

3 Protocol vulnerabilities

In this section we present a dictionary attack against the protocol proposed by Frikken *et al.* [9], which depends only on the temporary possession of the PUF and observation of a single trace of client/server communication. We also present a server impersonation attack against the protocol proposed by Busch *et al.* [8], which shows that the correction suggested by the authors is insufficient. Due to space constraints, we omit the protocol descriptions and refer the reader to the original versions [8,9], conserving most of the notation for compatibility.

3.1 Robust authentication using PUFs [9]

In their work, Frikken *et al.* [9] employ a zero-knowledge proof of possession of an I-PUF for client authentication through a bank-issued device and an additional password. The protocol does not provide mutual authentication between client and server, and does not establish a session key.

Since the challenge c is fixed, the PUF challenge $d = H(H(c||pwd), g, P)$ is also fixed, for some auxiliary information P. With possession of the PUF, the adversary computes a set $\mathcal{R}$ of values g^{r_i} corresponding to the PUF challenges $d_i = H(H(c||pwd_i), g, P)$, where pwd_i is a candidate password (this called a dictionary attack). Observe that the I-PUF assumption does not allow the attacker to obtain the responses r_i directly, but the result of the computation g^{r_i}, similar to the one performed during the enrollment phase, could be captured without the client's knowledge. Afterwards, the adversary replays to the client the challenge c, the group description $\langle \mathbb{G}_q, q \rangle$, auxiliary information P and nonce N observed in a previous honest communication. The user then sends $(H(c||pwd), \langle \mathbb{G}_q \rangle, q, P, N)$ to the device that computes $d = H(H(c||pwd), g, P)$ and executes the *Rep* procedure to obtain r. The device chooses a random value $v \in \mathbb{Z}_q$ and computes $t = g^v$. Following the protocol, the device computes $c = H(g, g^r, t, N)$ and $w = v - c'r \bmod q$, and returns c' and w to the client, which sends these values to the adversary. The adversary then calculates a set $\mathcal{T}$ of values $t_i = g^w g^{r_i c'}$, because he knows the values of g, w, c', and if $c' = H(g, g^{r_i}, t_i, N)$ for some value of i, learning therefore the user password if $pwd = pwd_i$.

Furthermore, an insider who steals the authentication records in this protocol [9] recovers the value g^r for some user being able to mount an offline attack by obtaining to the device and making queries to the PUF with several challenges $d_i = H(H(c||pwd_i), g, P)$ in hope of receiving $g^{r_i} = g^r$ as response. This observation directly contradicts a security claim against insider attacker stated in the Abstract and Introduction of the paper [9].

3.2 Strong authentication with PUFs [8]

Tuyls and Škorić proposed a lightweight PUF-based authentication protocol [7] providing session key establishment without a security analysis. This protocol is divided into two parts: the first is called the enrollment phase, in which an identifier ID_{PUF} is assigned to the PUF, a random sequence rd is chosen, and a set of challenges $\mathcal{C}$ is created. Then for each challenge $c_i \in \mathcal{C}$, the output s_i and auxiliary information ω_i are generated, forming the sets $\mathcal{S}$ and $\mathcal{W}$, respectively. The memory of the card is initialized with ID_{PUF}, $n = 0$ (number of previous authentication attempts), $m = rd$; and the server stores ID_{PUF}, $n = 0$, $m' = rd$ and $\{\mathcal{C}, \mathcal{W}, \mathcal{S}\}$ in a database.

In [8], Busch *et al.* suggested a different attacker model, introducing physical control of the PUF and of its respective reader to the attacker for a short time. This type of attacker is quite realistic in an authentication setting, for example, in the situation where the employee of a business establishment takes the client's credit card away from the client's view for billing. During this time the employee (adversary) can read data stored in its memory or perform some queries to obtain challenge-response pairs. Under this attacker model, an attack where the adversary $\mathcal{A}$ can impersonate the server by first choosing a small number of challenges $\mathcal{C}^*$ and calculating their respective responses $\mathcal{R}^*$. Afterwards, $\mathcal{A}$ reads the identifier ID_{PUF}, the usage counter n and the current hash value $h^n(m)$ that are stored on the memory of card. With this information, the adversary can compute the value $M^* = h^{n-n^*}(m)$ for $n^* > n$, since the counter n and the hash value $m = h(m)$ are directly stored in memory. Then $\mathcal{A}$ calculates $K_1^* = h(M^*||ID_{PUF})$, generates a random nonce β^* and chooses a challenge $c_i^* \in \mathcal{C}^*$ to generates its respective output s_i together with auxiliary information ω_i. Then $\mathcal{A}$ computes a MAC on $(\alpha||c_i^*||\omega_i^*||\beta^*)$ using the key K_1^*, encrypts the MAC with K_1^* and sends $Enc_{K_1^*}[(\alpha||c_i^*||\omega_i^*||\beta^*)||MAC_{K_1^*}(\alpha||c_i^*||\omega_i^*||\beta^*)]$ to the reader. The reader subsequently calculates $K_1 = (m||ID_{PUF})$, decrypts $Enc_{K_1^*}[(\alpha||c_i^*||\omega_i^*||\beta^*)||MAC_{K_1^*}(\alpha||c_i^*||\omega_i^*||\beta^*)]$ and verifies if the MAC is valid. Thus, since the MAC and decrypted nonce α are valid, the protocol does not abort with an error condition and a symmetric key K^* (respectively K) is established between the reader and the adversary, proving that the adversary can impersonate the server with success.

In order to mitigate this problem, the authors propose the use of Bloom filters [26] or hash trees [27] for storing in the card's memory the subset of challenges which were initially queried by the server. This storage is done compactly and does not allow an adversary with limited computational power to

gain useful information about the challenges. We shall see now that the additional storage overhead imposed on the client does not solve the vulnerability against server impersonation. Consider an adversary $\mathcal{A}$ who has access to the credit card including the PUF only once. With access to the card, $\mathcal{A}$ can read from the card's memory the identity ID_{PUF}, the usage counter n and the current value of hash of m. Then, the adversary establishes a honest connection with the server to determine α^*, n^*, ID_{PUF}, $Enc_{K_1^*}[T^*||MAC_{K_1^*}(T^*)]$, where $T^* = \alpha^*||c_i^{'*}||\omega_i^*||\beta^*$. Thus, $\mathcal{A}$ can find the challenge as follows: $\mathcal{A}$ calculates the value of $M^* = h^{n-n^*}(m)$, which is possible, because $\mathcal{A}$ has the usage counter n and the value of $m = H(m)$, computes $K_1^* = h(M^*||ID_{PUF})$ and can thus decrypt $Enc_{K_1^*}[T^*||MAC_{K_1^*}(T^*)]$. With the value of T^*, the adversary can obtain the c_i and ω_i that were used in the eavesdropped conversation. With a valid challenge c_i and its corresponding auxiliary information ω_i, the adversary can impersonate the server successfully in the same way by Busch *et al.* [8].

This attack works because the client does not verify the reuse of the same c_i. For this "verification" to work, the client must remove each c_i used from the card's memory from either the Bloom filter $\mathcal{B}$ or the hash tree, introducing additional complexity and storage costs [28].

4 PUF+PAKE protocol

The protocol presented in this section uses the combination of a PUF with a PAKE. The main idea is to use the PUF output as the shared password required by the PAKE. This protocol is divided into two phases: enrollment and authentication. In the enrollment phase, the server uses the PUF to generate tuples (c_i, ω_i, s_i) that will be used during the authentication phase. At this stage the server obtains a challenge d_i generated from a nonce c_i concatenated with the user's password pwd, and then uses the PUF output as input in the generation process Gen, presented in the Section 2.1, to obtain the output s_i and auxiliary information ω_i that are stored along with c_i. Notice that other authentication factors can be concatenated to c_i as input to the PUF, as suggested by Frikken *et al.* [9]. The server discards the tuple (c_i, s_i, ω_i) after using it in an authentication attempt.

The authentication phase is composed of three steps. In the first step, a shared secret is reproduced (which is the output of the PUF queried with the challenge d_i) between client and server. In the second step, a PAKE protocol satisfying the semantic secutity (freshness) and authentication properties is used to establish a new session key. Finally, in the third step, an additional key confirmation step is executed for ensuring mutual authentication, following the generic transform [17].

4.1 Enrollment phase

The enrollment phase of a client's device $\mathcal{D}$, performed according to Figure 1, is a step where the server generates and stores challenges for a client and their

corresponding responses. We consider $\mathcal{N}$ as a set of nonce $c_i \in [0,1]^{128}$, H as a hash function and $c_i \xleftarrow{R} \mathcal{N}$ indicates that c_i was sampled randomly from a set $\mathcal{N}$. We assume that this phase is **physically secured**, because in banking applications there is a trust relationship between client and bank, and the client can go the bank in person to participate in this phase. Each iteration of the enrollment phase generates a tuple (c_i, s_i, ω_i) in $\mathcal{C}$.

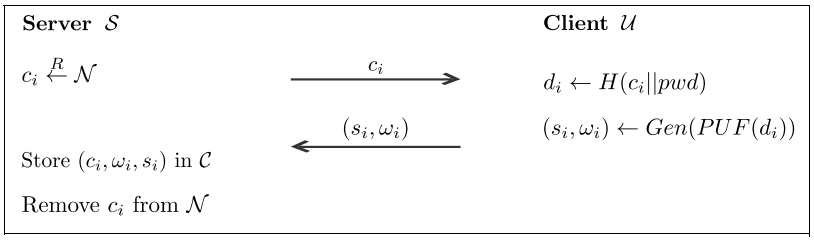

Fig. 1. Physically secure enrollment phase performed to generate a tuple $(c_i, s_i, \omega_i) \in \mathcal{C}$.

4.2 Authenticated key exchange phase

During the authentication, generation and verification of the session key are performed as in Figure 2. As mentioned before, this phase is divided into three steps, so the role of each feature, the PUF, the PAKE and the server authentication can be clear. The first part reproduces the value K_i and K_i' used in the combination between PUF and PAKE, in which the server randomly chooses a tuple from set $\mathcal{C}$, assigns s_i as value of his key K_i, sends c_i and ω_i to the client, which in turn, employs the PUF and password to generate the value for key K_i'. Notice that for the protocol to be performed successfully K_i must be equal to K_i'.

The second part is flexible and can be performed by any secure PAKE [15,16,20]. In Figure 2, we use an One-Encryption Key Exchange protocol (OEKE) [20,22], which is a simplified variant of the PAKE protocol proposed by Bellare *et al.* [17] with slight adaptations. This PAKE protocol has proofs of semantic security and authentication properties [20,22], that are the two necessary requirements for a PAKE protocol to be considered secure. Client authentication happens in this stage.

The third part is also flexible and there are other choices of protocols for key confirmation, as discussed by Jablon [16]. In this part, server authentication occurs and, hence, the mutual authentication between client and server is established. Functions H_0, H_1 represent cryptographic hash functions that can be constructed from a hash function H with different prefixes and E, D represent the encryption and decryption functions of an authenticated symmetric primitive (such as AES-GCM [29]), respectively.

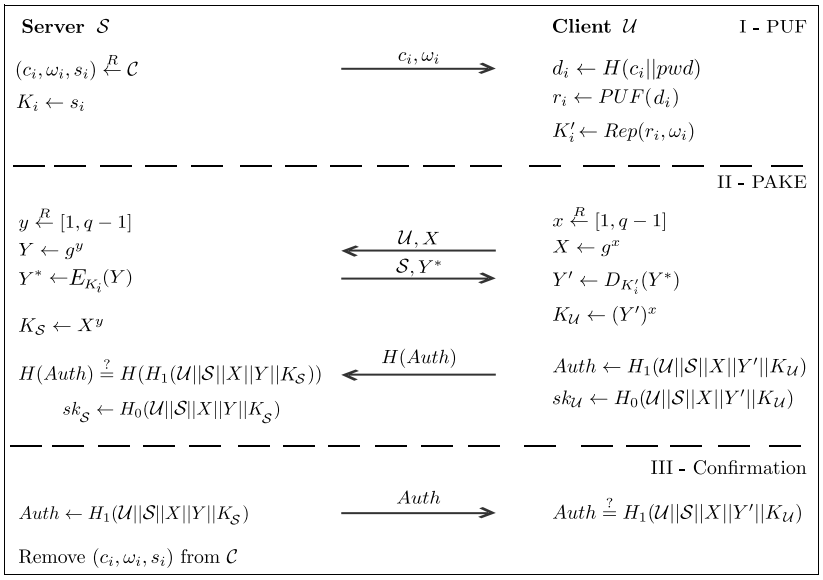

Fig. 2. Authenticated key exchange phase, divided into three parts, where we use the secure PAKE protocol.

5 Security

At any instant of time, the adversary does not have simultaneous access to all authentication factors, otherwise an attack becomes trivial. Thus, the adversary is unable to capture K_i directly. In order to prevent progressively leaking K_i by continuous interaction, a different tuple in $\mathcal{C}$ is used in each authentication attempt. We assume the bank has enough capacity for storing thousands of nonces c_i, and corresponding responses s_i with auxiliary information ω_i. This is different from other works that assume the existence of a stronger PUF inseparably bound to a chip able to perform computation (I-PUF) [30]. In this case, communication between the PUF and the chip is inaccessible to an attacker and thus cannot be tampered with.

Accordingly, we consider that the adversary $\mathcal{A}$ has temporary access to the PUF when he can perform a limited number of queries to build a set of PUF challenge-response pairs. If the number of authentication attempts is exceeded, the server can impose a limit of time that prevents an online exhaustive search attack in the password without blocking completely the client/server. The adversary also has access to network traffic between the client and server, corresponding to successful authentication attempts.

5.1 Security intuition

We analyze three scenarios in which the attacker tries to impersonate the client and the server.

Adversary does not have access to the PUF and to the password. In this scenario the adversary does not have access to PUF at any moment and does not know the client's password. The adversary only has access to previous honest communication traffic between the client and server.

- **Client and server**: since the PAKE is secure and satisfies the freshness property, the probability of an adversary impersonating the client or the server is the same probability as guessing K_i. For this to work, the adversary must be able to guess the response generated by the PUF, which happens with negligible probability, according to Section 2.1.

Adversary does have access to the PUF, but does not know the password. This scenario is the most realistic for authentication, where the adversary has access to the PUF for a limited time but does not have knowledge of the client's password *pwd*.

- **Client and server**: the adversary is able to impersonate the client successfully if he can guess the user's password. The probability for the adversary to impersonate the server successfully is considered negligible either if the distribution of passwords is nearly uniform or if there is a limited number of unsuccessful authentication attempts. Notice that the PAKE requires each authentication to involve interaction between client and server, allowing the client and the server to limit the number of unsuccessful attempts. Additionally, recall that the probability distribution of passwords is usually far from uniform, due to the fact that some passwords are commonly chosen and prone to dictionary attacks.

Adversary does not have access the PUF, but has access to the password. In this case the adversary cannot obtain PUF responses, but knows the client's password.

- **Client and server**: for the adversary to impersonate the client or the server, he needs to guess the output K_i of the PUF under input d_i. The probability of the adversary guessing this value is negligible, according to Section 2.1.

5.2 Formal analysis

A protocol for authenticated key exchange must satisfy two security notions: semantic security (also called freshness property in this context) and the authentication property. The first security notion ensures that the protocol produces a new shared cryptographic key as result. The second security notion ensures that the shared cryptographic key is obtained only for parties who meet certain authentication requirements.

Theorem 1 *The combined PUF+PAKE protocol using a semantically secure PAKE protocol remains semantically secure under the assumptions of the PAKE protocol and assuming a close to uniform statistical distribution of the PUF outputs.*

Proof. A PAKE protocol is semantically secure given a set of computational assumptions and uniformly random choice of a shared password. A PUF modeled as an unpredictable function postprocessed by a fuzzy extractor satisfies this requirement. Therefore, the combined construction only transfers the semantic security and other security properties from the PAKE protocol to the PAKE+PUF protocol under the same computational assumptions that guarantee the semantic security of the PAKE protocol isolated.

Theorem 2 *A polynomial adversary with access to the PUF (with security parameter ℓ) has negligible probability of success in the PUF+PAKE authentication protocol for a previous user enrollment, assuming that H is a random oracle, the PAKE protocol satisfies the property of authentication and passwords are chosen from a set large enough for the probability of guessing to be also negligible.*

Proof. This security reduction is an adaptation of Theorem 1 as seen in the work from [9]. Assuming that an adversary $\mathcal{A}$ with non-negligible success probability exists for the authentication property, we construct an adversary $\mathcal{B}$ for the PUF indistinguishability using $\mathcal{A}$ as a black box. The adversary $\mathcal{B}$ chooses a random challenge c and a random password pwd, computes $d = H(c||pwd)$ and chooses d as his PUF challenge for the game of indistinguishability. The adversary $\mathcal{B}$ receives a pair (r_b, ω) determined by the random bit b such that $r_0 = Rep(PUF(d), \omega)$ and r_1 is randomly chosen; and instantiates the adversary $\mathcal{A}$ with values (c, ω) giving oracle access to H and PUF. To simulate the random oracle H, $\mathcal{B}$ creates a set of tuples initialized as $H_S = (c||pwd, h)$ for a randomly chosen value h. When $\mathcal{A}$ queries the oracle with input x, $\mathcal{B}$ verifies if a pair (x, y) already exists in H_S and returns y, otherwise adds (x, h') to set H_S and returns the randomly chosen value h' as result. To simulate the PUF for a query (d', ω'), $\mathcal{B}$ checks if $d = d'$ and returns FAIL if positive. Otherwise, $\mathcal{B}$ returns $(r', \omega') = Gen(d')$ as a result. Thus, $\mathcal{A}$ has a view indistinguishable from the real protocol and eventually produces a proof of authentication for the possibly shared key. If this proof of authentication is correct, $\mathcal{B}$ returns 0 as a result, or an random bit b' otherwise.

First, let's analyze the probability $\Pr[b = b']$. Let F be the event that $\mathcal{B}$ returns FAIL. This event occurs with only negligible probability, since it requires that the adversary $\mathcal{A}$ guess the password pwd or the PUF challenge d to query the PUF. One can divide the remaining case $\Pr[b = b'|\overline{F}]$ for the two values of b:

$$\Pr[b = b'|\overline{F}] = \frac{1}{2}\Pr[b = b'|\overline{F}, b = 0] + \frac{1}{2}\Pr[b = b'|\overline{F}, b = 1].$$

Let G be the event that $\mathcal{A}$ produces a correct proof of authentication. We condition both of the above cases on G. For the case $b = 1$:

$$\Pr[b = b'|\overline{F}, b = 1] = \Pr[b = b'|\overline{F}, b = 1, G]\Pr[G|\overline{F}, b = 1] + \Pr[b = b'|\overline{F}, b = 1, \overline{G}]\Pr[\overline{G}|\overline{F}, b = 1].$$

We have that $\Pr[b = b'|\overline{F}, b = 1, G] = 0$, because $b = 0$ for the event G; $\Pr[b = b'|\overline{F}, b = 1, \overline{G}] = \frac{1}{2}$, because $b = 1$ occurs with 50% for the event $\overline{G}$, and

$\Pr[G|\overline{F}, b = 1]$ is a negligible function by the security of the PAKE protocol. Hence, for some negligible function $\lambda(\ell)$:

$$\Pr[b = b'|\overline{F}, b = 1] > \frac{1}{2} - \lambda(\ell).$$

The case $b = 0$ is similar:

$$\Pr[b = b'|\overline{F}, b = 0] = \Pr[b = b'|\overline{F}, b = 0, G] \Pr[G|\overline{F}, b = 0] + \Pr[b = b'|\overline{F}, b = 0, \overline{G}] \Pr[\overline{G}|\overline{F}, b = 0].$$

Here, $\Pr[b = b'|\overline{F}, b = 0, G] = 1, \Pr[b = b'|\overline{F}, b = 0, \overline{G}] = \frac{1}{2}$ and $\Pr[G|\overline{F}, b = 0] > \frac{1}{f(\ell)}$ for some polynomial f, by the assumption that $\mathcal{A}$ breaks the authentication property with non-negligible probability. Therefore:

$$\Pr[b = b'|\overline{F}, b = 0] > \frac{1}{f(\ell)} + \frac{1}{2} \cdot \left(1 - \frac{1}{f(\ell)}\right) = \frac{1}{2} + \frac{1}{2f(\ell)}.$$

Substituting the terms, we have:

$$\Pr[b = b'|\overline{F}] > \frac{1}{2}\left(\frac{1}{2} + \frac{1}{2f(\ell)}\right) + \frac{1}{2}\left(\frac{1}{2} - \lambda(\ell)\right).$$

In summary, if we have $\Pr[b = b'|\overline{F}] - \frac{1}{2}$ non-negligible then, $\Pr[b = b'] - \frac{1}{2}$ is also non-negligible and the attacker $\mathcal{A}$ must exist, contradicting the hypothesis that the PAKE protocol is secure.

5.3 Checklist analysis

Delvaux *et al.* [6] were the pioneers in enumerating a set of ten requirements that PUF-based protocols should possess. In the following, we do a brief analysis about how our protocol fits into each one these requirements.

1 **Complete specification**: the PUF+PAKE protocol has a complete unambiguous specification with graphical representation of both enrollment and authentication phases, showing details of all computations and exchanged messages between client and server. Also, we loosely recommend a PUF [13], noting that the protocol can be used with any PUF satisfying the security properties.
2 **Leakage resilience**: the PUF+PAKE protocol does not impose secure data storage on the client, hence, leakage of information does not occur. Data is only stored in server side in a non-volatile memory that is assumed secure.
3 **Able to handle noisiness**: the PUF+PAKE protocol handles noisiness and also non-uniform distribution of PUF outputs, using a fuzzy extractor, as shown in Section 2.1
4 **Counteracting strong PUF modeling attacks**: the adversary is unable to capture PUF challenges, because the challenges d_i are not transmitted or stored in the server. The challenges are correctly generated only if the adversary knows the user's password. Hence, collecting pairs (d_i, s_i) for mounting machine learning modeling attacks requires physical access to the PUF.

5 **Strong PUF**: the PUF+PAKE protocol requires a strong PUF providing a large set of challenges. Hence, we can use any PUF that has response space expansion [13].
6 **Low-cost and resource-constrained**: the PUF+PAKE protocol only requires symmetric primitives and few public key operations. The latter can be instantiated with elliptic curves to benefit from efficient implementations and reduced parameter sizes.
7 **Easy-to-instantiate**: the PUF+PAKE protocol is easy-to-instantiate because we designed the protocol to not depend on a specific PUF, but any PUF that satisfies the security requirements.
8 **Resistance against protocols attacks**: the PUF+PAKE protocol has a formal analysis of security (Section 5.2) and no attacks are known.
9 **Scalability:** in banking applications, identification is easy and clients provide the bank branch and account number before performing the authentication protocol.
10 **On the Mutual Authentication Order**: in PUF+PAKE client authentication happens first, because in banking applications client impersonation is more common. Changing the order of the messages at the end of the second part (PAKE) is enough to adapt the protocol for applications in which the server authenticates first.

6 Emergency

Panic passwords are mechanisms that allow users to use a special kind of password, called *panic password*, to signal the server (or any other communicant party) that his password is being inserted as a result of coercive action [31]. They are also known as help passwords or codes in the literature. A popular example of the use of panic passwords was employed in older versions of the RSA SecurID device [32]. This device is intended to perform two-factor user authentication for accessing network resources using both a personal identification number and a one-time password generated by the token.

The protocol shown in Figure 2 can easily support panic passwords [31], by using an adaptation of a known technique [9]. There are two valid passwords for each user, both with a fixed-length shared prefix (pwd_1) and distinct suffixes that indicate normal situation (pwd_2) and emergency (pwd_3). Let pwd^* represent one of the two valid suffixes. The server is able to distinguish between the two situations by checking what session key was derived by the protocol execution. The enrollment phase undergoes some changes and proceeds as shown in Figure 3.

In the authentication phase, the roles for the client and server are reversed in the illustrative PAKE, which causes no security impact when the generic transformation of mutual authentication is applied [17]. This reversion is needed to transfer the encryption step to the client, which will send the encryption under one of the two possible keys, indicating normal or emergency situation. The server can then decrypt the received message and check what kind of situation the next message from the client corresponds.

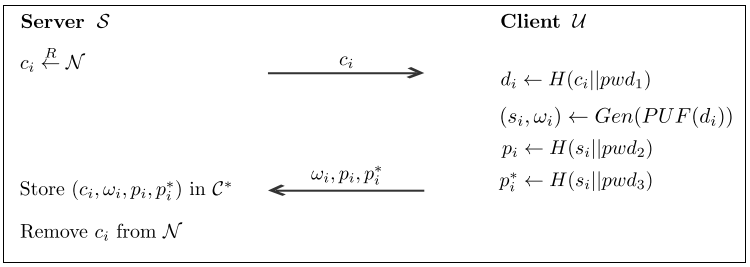

Fig. 3. Physically secure panic enrollment phase performed to generate a tuple $(c_i, \omega_i, p_i, p_i^*) \in \mathcal{C}^*$.

When the server detects an emergency situation, it executes the protocol normally, so that the adversary does not notice that the panic password has been used, therefore, both executions must be *indistinguishable*. In this case, the server can send a signal to the police, warning that something suspicious is happening or even limit the amount of money available for withdrawal.

In most cases, when there is no emergency situation, the server continues normally and sends its message to mutual authentication. The resulting protocol is described in Figure 4. Naturally, when notification of emergency is desirable, all clients in any situation should use the adjusted protocol so that normal instances of the protocol have the same communication pattern as emergency situations. The formal analysis of security of this variant follows directly from the formal analysis of the Section 5.2, since only simple modifications were required in the protocol.

7 Transaction authentication protocol

In banking applications, ensuring that both parties, server and client, are authenticated in the beginning of a session is not enough. It is also important to ensure that the client remains connected and with access to the PUF at the time when transactions are performed. Protocols with this feature are calls transaction authentication protocols. The Figure 5 shows our transaction authentication protocol, to be executed for each new transaction in a session. The protocol is linked to the PUF+PAKE protocol shown in Section 4, since the same session key is used here ($sk_{\mathcal{S}}$ and $sk_{\mathcal{U}}$) to ensure that the transaction is indeed occurring in the current session. For this reason, it is a two-factor transaction authentication protocol, implicitly authenticating participation in the current session and access to the previously enrolled PUF.

The protocol is performed as follows: first, the client encrypts the transaction data T with the session key ($sk_{\mathcal{U}}$) and then sends it to the server. The server then decrypts the transaction with its session key ($sk_{\mathcal{S}}$), randomly chooses a tuple on the set $\mathcal{C}$ and generates a new key K_j based on the concatenation of the PUF output (for a new nonce c_j and consequently a new d_j to the client)

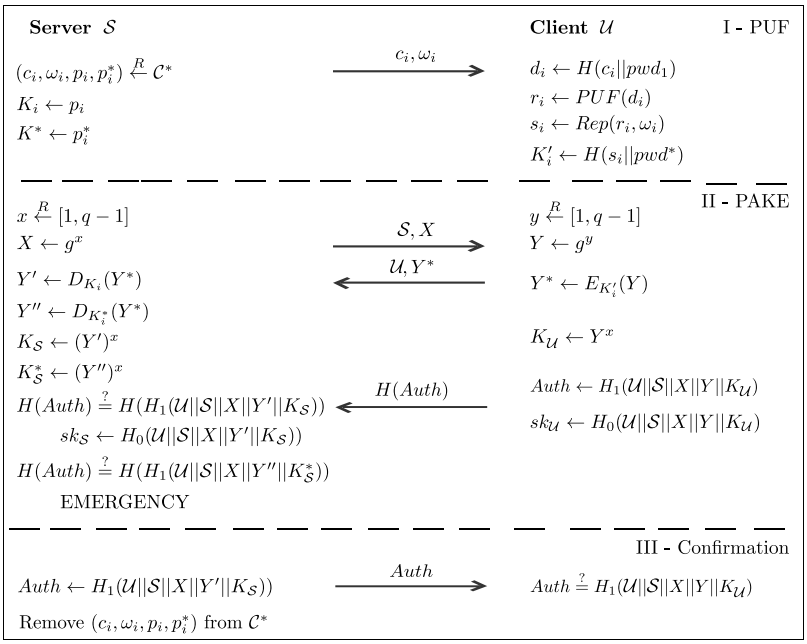

Fig. 4. Authenticated key exchange phase with support for panic passwords, where pwd^* represent one of the two valid suffixes (normal (pwd_2) or emergency (pwd_3) situation).

and the session key. The server then chooses a 4-digit one-time password z, that is concatenated with T' as $(z||T')$ and then encrypted using K_j and sent together with the nonce c_j and the auxiliary information ω_j. The client compares the received transaction data with the requested transaction, and if positive calculates the value of K_j' using the PUF, his password pwd and the session key $sk_{\mathcal{U}}$. Then, the client decrypts the message Z^* and returns the decrypted one-time password for verification[2]. Finally, the server remove the tuple (c_i, ω_i, s_i) from $\mathcal{C}$.

To reduce the size of $\mathcal{C}$, one can employ the same tuple (c_i, ω_i, s_i) on all transactions in the same session, one for entity authentication and another for transaction authentication. Because of this, only two tuples of $\mathcal{C}$ are necessary per session. In the Figure 5 , E and D are defined as previously.

7.1 Security analysis

The security analysis is restricted to the scenario where the adversary does not have access to the PUF. Consider that the PUF+PAKE protocol was performed

[2] This is the only user interaction with the protocol, besides making the transaction request. The other operations are done by *software.*

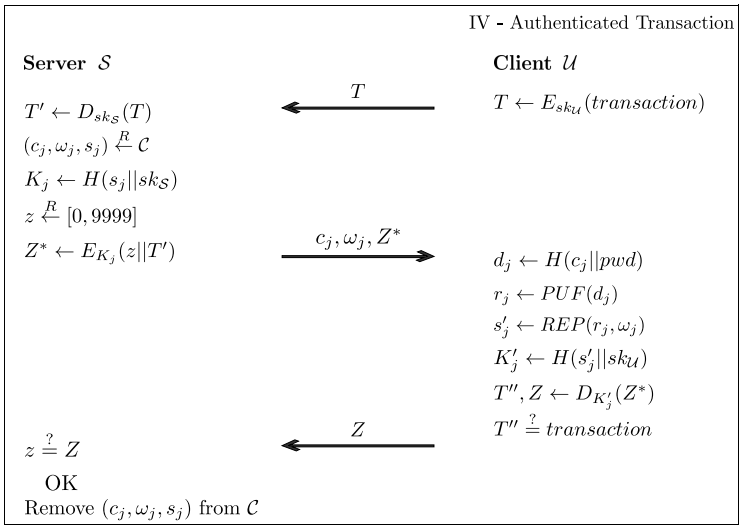

Fig. 5. Transaction authentication protocol that use the same tuple (c_i, ω_i, s_i) on all transactions in the same session.

successfully and securely, and consequently the session key is only known by authenticated parties.

Assume that the adversary does not have access to the PUF and/or user's password at any moment. Otherwise, and if the adversary had authenticated in the PUF+PAKE protocol, an attack is trivial. The adversary thus only has access to previous traffic of honest communications between the client and server.

For an adversary to impersonate the client successfully, he would have to guess the values of $sk_{\mathcal{U}}$ (client's session key) and the output of the PUF for the challenge d_j that can only be computed if the adversary knows the client's password. As the PUF+PAKE protocol is secure, the advantage of the adversary in guessing $sk_{\mathcal{S}}$ is negligible and the probability of guessing the output of the PUF, according to Section 2.1, is also negligible. Thus, the advantage of the adversary successfully impersonating the client is negligible.

8 Comparison

Considering the scenarios discussed in Section 5.1 the protocol developed by Frikken *et al.* [9] has the same security properties as the protocol proposed in this paper, with a notable exception when the adversary has temporary access to the PUF, but does not know the user's password. In the PUF+PAKE protocol, the adversary successfully impersonates the client or the server under these circumstances with negligible probability for each authentication attempt. In Section 3.1, we presented a server impersonation attack that allows the server to discover the user password with higher probability. This happens because the

adversary only needs to impersonate the server one time to check if the user password is included in the set of candidate passwords queried through the PUF and the remaining work can be done offline. The PAKE in our protocol forces the adversary to impersonate the client/server to check a single password candidate per authentication attempt, further allowing either the client or the server to monitor the malicious behavior of the other party. Comparing our protocol to Frikken *et al.* provides an interesting security trade-off: while our protocol provides stronger resistance against dictionary attacks, an insider attack is easier to mount in PUF+PAKE because the PUF response is stored in the server.

Delvaux *et al.* [6] present many attacks against lightweight authentication protocols. Our attack in Section 3.2 against the protocol proposed by Tuyls and Škorić [7] is new, and illustrates the inherent limitations of lightweight protocols for our target application. In short, lightweight protocols are not designed to receive and protect additional credentials (multifactor feature) against offline attacks. For attaining this goal, more computationally expensive protocols forcing client-server interaction in every authentication attempt are needed.

9 Conclusions and future work

In this paper, we proposed a flexible authentication protocol based on a combination of PUF with PAKE that provides mutual authentication between the client and the server establishing a session key, some of the main features for a useful authentication protocol. This protocol can be used in various environments, for example authentication in banking applications, in which the session key can be used to authenticate subsequent transactions or protect financial information in transit. In particular, the PUF+PAKE combination improves the state of the art of authentication solutions based on PUFs, according to the formal analysis presented. Additionally, a variant of the protocol is proposed to support panic passwords for emergency situations where the client is compelled to deliver the PUF and reveal his password, and a two-factor authentication solution for transaction authentication is discussed. We also presented server impersonation attacks on two PUF-based authentication protocols proposed in the literature, motivating the need for mutual authentication in such applications.

For future work, we plan to develop an alternate protocol that provides security against both insider agents and offline dictionary attacks, satisfying both of the security properties. Finally, it is important to implement the proposal with a real PUF candidate for obtaining performance/reliability measures and studying its practical feasibility.

References

1. R. Pappu, B. Recht, J. Taylor, and N. Gershenfeld, "Physical One-Way Functions," *Science*, vol. 97, pp. 2026–2030, 2002.
2. B. Gassend, D. Clarke, M. van Dijk, and S. Devadas, "Silicon Physical Random Functions," in *Proceedings of the 9th ACM Conference on Computer and Communications Security (CCS 2002)*, (New York, NY, USA), pp. 148–160, ACM, 2002.

3. F. Armknecht, R. Maes, A.-R. Sadeghi, B. Sunar, and P. Tuyls, "Memory Leakage-Resilient Encryption Based on Physically Unclonable Functions," in *15th International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT 2009)* (M. Matsui, ed.), vol. 5912 of *Lecture Notes in Computer Science*, pp. 685–702, Springer Berlin Heidelberg, 2009.
4. G. E. Suh and S. Devadas, "Physical Unclonable Functions for Device Authentication and Secret Key Generation," in *Proceedings of the 44th Annual Design Automation Conference (DAC 2007)*, (New York, NY, USA), pp. 9–14, ACM, 2007.
5. R. Maes, A. Van Herrewege, and I. Verbauwhede, "PUFKY: A Fully Functional PUF-Based Cryptographic Key Generator," in *Cryptographic Hardware and Embedded Systems (CHES 2012)* (E. Prouff and P. Schaumont, eds.), vol. 7428 of *Lecture Notes in Computer Science*, pp. 302–319, Springer Berlin Heidelberg, 2012.
6. J. Delvaux, D. Gu, R. Peeters, and I. Verbauwhede, "A Survey on Lightweight Entity Authentication with Strong PUFs." COSIC Internal Report (`http://www.cosic.esat.kuleuven.be/publications/article-2497.pdf`), 2015.
7. P. Tuyls and B. Škorić, "Strong Authentication with Physical Unclonable Functions," in *Security, Privacy, and Trust in Modern Data Management* (M. Petković and W. Jonker, eds.), Data-Centric Systems and Applications, pp. 133–148, Springer Berlin Heidelberg, 2007.
8. H. Busch, S. Katzenbeisser, and P. Baecher, "PUF-Based Authentication Protocols - Revisited," in *Information Security Applications* (H. Youm and M. Yung, eds.), vol. 5932 of *Lecture Notes in Computer Science*, pp. 296–308, Springer Berlin Heidelberg, 2009.
9. K. Frikken, M. Blanton, and M. Atallah, "Robust Authentication Using Physically Unclonable Functions," in *Information Security* (P. Samarati, M. Yung, F. Martinelli, and C. Ardagna, eds.), vol. 5735 of *Lecture Notes in Computer Science*, pp. 262–277, Springer Berlin Heidelberg, 2009.
10. R. Maes, "An Accurate Probabilistic Reliability Model for Silicon PUFs," in *Cryptographic Hardware and Embedded Systems (CHES 2013)* (G. Bertoni and J.-S. Coron, eds.), vol. 8086 of *Lecture Notes in Computer Science*, pp. 73–89, Springer Berlin Heidelberg, 2013.
11. C. Helfmeier, C. Boit, D. Nedospasov, and J.-P. Seifert, "Cloning Physically Unclonable Functions," in *International Symposium on Hardware-Oriented Security and Trust (HOST 2013)*, pp. 1–6, IEEE, June 2013.
12. S. Katzenbeisser, Ü. Koçabas, V. Rozic, A. Sadeghi, I. Verbauwhede, and C. Wachsmann, "PUFs: Myth, Fact or Busted? A Security Evaluation of Physically Unclonable Functions (PUFs) Cast in Silicon," in *Cryptographic Hardware and Embedded Systems (CHES 2012)* (E. Prouff and P. Schaumont, eds.), vol. 7428 of *Lecture Notes in Computer Science*, pp. 283–301, Springer Berlin Heidelberg, 2012.
13. D. Holcomb and K. Fu, "Bitline PUF: Building Native Challenge-Response PUF Capability into Any SRAM," in *Cryptographic Hardware and Embedded Systems (CHES 2014)* (L. Batina and M. Robshaw, eds.), vol. 8731 of *Lecture Notes in Computer Science*, pp. 510–526, Springer Berlin Heidelberg, 2014.
14. Y. Dodis, L. Reyzin, and A. Smith, "Fuzzy Extractors: How to Generate Strong Keys from Biometrics and Other Noisy Data," in *Advances in Cryptology (EUROCRYPT 2004)* (C. Cachin and J. Camenisch, eds.), vol. 3027 of *Lecture Notes in Computer Science*, pp. 523–540, Springer Berlin Heidelberg, 2004.
15. S. M. Bellovin and M. Merritt, "Encrypted Key Exchange: Password-based Protocols Secure Against Dictionary Attacks," in *Research in Security and Privacy,*

1992. Proceedings., 1992 IEEE Computer Society Symposium on, pp. 72–84, IEEE, May 1992.
16. D. P. Jablon, "Strong Password-only Authenticated Key Exchange," *ACM SIGCOMM Computer Communication Review*, vol. 26, pp. 5–26, Oct. 1996.
17. M. Bellare and P. Rogaway, "The AuthA Protocol for Password-based Authenticated Key Exchange," tech. rep., Citeseer, 2000.
18. V. Boykoy, P. MacKenziez, and S. Patelx, "Provably Secure Password-Authenticated Key Exchange Using Diffie-Hellman," 2000.
19. J. Katz, R. Ostrovsky, and M. Yung, "Efficient Password-Authenticated Key Exchange Using Human-Memorable Passwords," in *Advances in Cryptology (EUROCRYPT 2001)* (B. Pfitzmann, ed.), vol. 2045 of *Lecture Notes in Computer Science*, pp. 475–494, Springer Berlin Heidelberg, 2001.
20. E. Bresson, O. Chevassut, and D. Pointcheval, "Proof of Security for Password-based Key Exchange (IEEE P1363 AuthA Protocol and Extensions)," *ACMCCS*, vol. 3, pp. 241–250, 2003.
21. M. Abdalla, D. Catalano, C. Chevalier, and D. Pointcheval, "Efficient Two-Party Password-Based Key Exchange Protocols in the UC Framework," in *Topics in Cryptology (CT-RSA 2008)* (T. Malkin, ed.), vol. 4964 of *Lecture Notes in Computer Science*, pp. 335–351, Springer Berlin Heidelberg, 2008.
22. E. Bresson, O. Chevassut, and D. Pointcheval, "Security Proofs for an Efficient Password-based Key Exchange," in *Proceedings of the 10th ACM Conference on Computer and Communications Security (CCS 2003)*, (New York, NY, USA), pp. 241–250, ACM, 2003.
23. S. P. Miller, B. C. Neuman, J. I. Schiller, and J. H. Saltzer, "Kerberos Authentication and Authorization System," in *In Project Athena Technical Plan*, Citeseer, 1987.
24. J. G. Steiner, B. C. Neuman, and J. I. Schiller, "Kerberos: An Authentication Service for Open Network Systems," in *USENIX Winter*, pp. 191–202, 1988.
25. K.-Y. Lam and D. Gollmann, "Freshness Assurance of Authentication Protocols," in *2th European Symposium on Research in Computer Security (ESORICS 1992)* (Y. Deswarte, G. Eizenberg, and J.-J. Quisquater, eds.), vol. 648 of *Lecture Notes in Computer Science*, pp. 261–271, Springer Berlin Heidelberg, 1992.
26. B. H. Bloom, "Space/Time Trade-offs in Hash Coding with Allowable Errors," *Communications of the ACM*, vol. 13, pp. 422–426, July 1970.
27. R. C. Merkle, "Protocols for Public Key Cryptosystems.," in *IEEE Symposium on Security and privacy*, vol. 1109, pp. 122–134, 1980.
28. L. Fan, P. Cao, J. Almeida, and A. Z. Broder, "Summary Cache: A Scalable Wide-area Web Cache Sharing Protocol," *IEEE/ACM Transactions on Networking (TON)*, vol. 8, pp. 281–293, June 2000.
29. M. J. Dworkin, "SP 800-38D. Recommendation for block cipher modes of operation: Galois/Counter Mode (GCM) and GMAC," 2007.
30. P. Tuyls and L. Batina, "RFID-Tags for Anti-counterfeiting," in *The Cryptographers' Track at the RSA Conference (CT-RSA 2006)* (D. Pointcheval, ed.), vol. 3860 of *Lecture Notes in Computer Science*, pp. 115–131, Springer Berlin Heidelberg, 2006.
31. J. Clark and U. Hengartner, "Panic Passwords: Authenticating under Duress," *HotSec*, vol. 8, p. 8, 2008.
32. N. Popp, S. Bajaj, and P. Hallam-Baker, "Hybrid authentication," jan 2005. US Patent App. 10/864,501.