2017 IEEE 30th Computer Security Foundations Symposium

Human Computing for Handling Strong Corruptions in Authenticated Key Exchange

Alexandra Boldyreva and Shan Chen
School of Computer Science
Georgia Institute of Technology
266 Ferst Dr., Atlanta, GA 30332, USA
{sasha, shanchen}@gatech.edu

Pierre-Alain Dupont and David Pointcheval
Département d'informatique de l'ENS
École Normale Supérieure, CNRS
PSL Research University, 75005 Paris, France
and INRIA
{pierre-alain.dupont, david.pointcheval}@ens.fr

***Abstract*—We propose the first user authentication and key exchange protocols that can tolerate strong corruptions on the client-side. If a user happens to log in to a server from a terminal that has been fully compromised, then the other past and future user's sessions initiated from honest terminals stay secure. We define the security model for Human Authenticated Key Exchange (HAKE) protocols and first propose two generic protocols based on human-compatible (HC) function family, password-authenticated key exchange (PAKE), commitment, and authenticated encryption. We prove our HAKE protocols secure under reasonable assumptions and discuss efficient instantiations. We thereafter propose a variant where the human gets help from a small device such as RSA SecurID. This permits to implement an HC function family with stronger security and thus allows to weaken required assumptions on the PAKE. This leads to the very efficient HAKE which is still secure in case of strong corruptions. We believe that our work will promote further developments in the area of human-oriented cryptography.**

***Index terms*—Human computation, key exchange, one-time passwords, PAKE, strong corruptions.**

I. Introduction

MOTIVATION AND FOCUS. Consider a very common scenario when a user needs to log in to and securely communicate to a server, with which she shares a secret. This problem has been extensively studied under the name of Password-Authenticated Key Exchange (or PAKE), since the seminal paper by Bellovin and Merritt [1]. But what happens if the client terminal the user logs in from has been compromised? The machine may have a spyware keylogger recording the user's keystrokes and sending them to the attacker. A session hijacking malware may alter the legitimate computation and impersonate the user or the server.

The existing security definitions for PAKE acknowledge the problem by modeling *strong* corruptions when the adversary learns all the current state of the machine. However, none of the existing protocols try to offer any solution in this case. Basically, the consensus is that in case of strong corruption, all is lost to the user, and the only thing guaranteed is that this should not violate security of other users. Indeed, cryptography cannot do much since the attacker invading a machine would know everything, as it can read all secrets being stored or typed.

In this paper, we take a fresh look at this problem of strong corruptions with the intention of providing a solution. The informal goal is as follows. Given fully untrusted machines, a user's sessions (past and future) from other trusted terminals are still protected, even though the same long-term secret is used. As we said, it seems like nothing can be done cryptographically. But there are possibilities. The basic idea is to store no long-term secrets on the machines, and instead, employ human computation or an additional secure device such as RSA SecurID to boost security. (We think it is much more reasonable to assume that the human or the small device not connected to a network stays uncompromised, than terminals and other devices used for connecting to servers.) In a bit more detail, we ask the human user to log in by computing (in her head or with an additional device) a function of the memorized long-term secret and a challenge sent by the server, and entering it into the terminal. Then we can use a PAKE-like protocol ran on the response as a common ephemeral secret, also known as a *one-time password*.

A PAKE, that is usually used to prevent off-line dictionary attacks, here provides the guarantee that no information is leaked about the one-time passwords in passive and even active sessions. It is important to limit the information leakage about the long-term secret of the user, since one-time passwords, were they in the clear, could have helped recovering the long-term secret. This is unfortunately the case when they are generated with functions that are easy enough to be computed by a human. On the other hand, if an additional device is used to derive the one-time passwords, their privacy may be less critical, and so resistance to off-line dictionary attacks is not required anymore, which allows the use of a weaker variant of PAKE. To make these ideas "work", numerous problems need to be resolved to finalize the solutions. We discuss these after we describe our security model.

PROTOCOL AND SECURITY DEFINITIONS. The novelty behind our definitions is the unavoidable incorporation of a human player. We define a *human authenticated key exchange* (or HAKE) protocol as an interactive protocol between a human user U and a server S, via a terminal T. The server can only directly communicate with the terminal, and the user can only directly communicate with the terminal. In addition,

DOI 10.1109/CSF.2017.31

the messages sent by the terminal to the user must be *human-readable*, the messages sent by the user to the terminal must be *human-writable*, and the long-term secret of the user must be *human-memorizable*, unless an additional device is used for computing the ephemeral secrets.

Our security model is a non-trivial extension of the security model for PAKE protocols by Bellare, Pointcheval, and Rogaway [2], later called BPR, as we take into account really strong corruptions and model human computations/interactions. As already mentioned, the goal of a HAKE protocol is to ensure that a human user sharing the long-term secret with a server can establish a secure channel with the server, in presence of a very strong attacker. Our model takes into account various types of attacks possible in practice. As usual, to model a network compromise (e.g., taking advantage of an insecure Wi-Fi), we allow the adversary to control the messages parties exchange. The attacker can read and modify the communication between a server and a terminal. However, we assume that the channel between the human user and the *honest* terminal is secure, since this is a direct communication from the keyboard and the screen. At least, it is authentic and private, unless the terminal is compromised.

We thus also have to model malicious terminals and this, in fact, is the gist of our work. Even though taking the full control of a computer is an extremely hard job, compromising its parts, such as a browser, is very common. And such compromises can be of various strengths. Using a keylogger, screen capture or similar malware the adversary can learn the terminal's inputs/outputs. The attacker may also learn some random coins or intermediate values from the internal state of the compromised computer. This also models human "over-the-shoulder" attacks. Even though such compromise can be referred to as honest-but-curious, the existing protocols, such as PAKE, do not offer protection against it. The existing protocols only protect against *weak* corruptions, where the attacker just learns the session key (with reveal-queries). This models the misuse of the session key, rather than the terminal compromise. Even if security models for PAKE allow the attacker to learn the long-term secrets [2] (with corrupt-queries), or the internal states (in the UC framework [3]), this is only to model forward secrecy, and so the security of past sessions, but nothing is guaranteed anymore for future sessions.

In our model, we let the adversary compromise terminals and learn all their inputs and the internal state. Moreover, we consider an even more powerful adversary, who takes full control of the terminal's browser and can display outputs of its choice to be shown to the human user. Hence, the adversary can interact with the human user, but is never given the long-term secret key memorized by the human user (or stored on her secondary device).

The security goals are, to the most part, the standard privacy and authentication for key exchange protocols: we want to make sure that an attacker cannot learn any information about the session key nor make a party agree on a session key without the other party completing the protocol. Of course, if a terminal is compromised, it is unreasonable to expect security of the current session. But this should not compromise security of other sessions (past or future), even involving the same user.

HAKE PROTOCOL: GENERIC CONSTRUCTIONS AND INSTANTIATIONS. Let us assume we have a human-compatible function family F (we will discuss it in more detail shortly). Let the server pick a random challenge x (or increment a counter) and display it to the human user via the user's terminal. The user can compute (in her head or using a device) and enter the response $r = F_K(x)$, where K is the long-term secret shared between the user and the server. The server can compute r on its end the same way. Then, the terminal and the server execute a PAKE protocol on r (i.e., the response r plays the role of the password in PAKE), and thus agree on a session key. Even though human-computable responses may have low entropy, PAKE ensures security against off-line dictionary attacks, which guarantees no information leakage about the ephemeral secret r in passive sessions, and even in active sessions, excepted possibly the exclusion of one candidate per session. If the attacker compromises the terminal, a suitable "unforgeability" property of F would prevent the adversary from breaking security of other sessions.

But still, this protocol is not secure under our definition. An attacker, who learns an ephemeral secret $r = F_K(x)$ for a given challenge x can later use it to successfully impersonate the server, by forcing the same challenge. To prevent such replay attacks, we let the terminal and the server to jointly pick a challenge using a coin-flipping protocol, that we implement using a commitment scheme with specific properties. This is our first proposal, which we call the *Basic HAKE*: using a coin-flipping, we avoid replay attacks, and with a suitable unforgeability' property on the function family F we can guarantee the security of the global process.

However, a malicious terminal can still ask specific (not necessarily random) challenges to the human user, while impersonating the server, and the user has no way to detect such a malicious behavior. Therefore security of the Basic HAKE requires that the HC function unforgeability holds even in presence of multiple adaptive challenges. This may be too strong of a requirement in practice. We thereafter enhance *Basic HAKE* and propose the *Confirmed HAKE* protocol, which allows parties to detect potential bad behaviors, in order to react appropriately, and thus the construction tolerates weaker HC function families. Requirements on HC function families then become more compatible with functions that can be evaluated by human being without external help. The Confirmed HAKE also provides explicit authentication of the parties.

Finally, we consider the case of a device-assisted protocol: with such an additional device, one can implement more complex computations, and thus use stronger HC function families. This leads to less critical ephemeral secrets: leaking information about several $(x, F_K(x))$ pairs might not endanger the long-term secret K. This allows us to rely on a weaker variant of PAKE, and hence get a device-assisted HAKE that

is more efficient.

HUMAN-COMPATIBLE FUNCTION FAMILY. We now turn our attention to the inner part of the construction, the human-compatible function family. Security-wise, the adversary should be able to see multiple challenge-response pairs, among which some of the challenges could be chosen by the attacker (adaptive queries vs. non-adaptive queries). This is because the attacker, who compromises a terminal, can eavesdrop on the communication with the human user. And an active attacker who took control of the terminal can impersonate the server and ask the user to answer maliciously chosen challenges. But still, the adversary should not be able to forge a valid response for a new random challenge, so that future sessions remain safe.

Finding such a function family would be easy if we did not have the human-computability restrictions. We survey some works on secure human-based computation later, but they are not directly suitable for us. Luckily, a recent paper "Towards human computable passwords" by Blocki, Blum, Datta, and Vempala [4] (almost) provides a solution. The paper proposes a way for a human user to authenticate to a computer that does not offer privacy (honest-but-curious). Such a computer stores a set of challenges and the user authenticates by providing a response to a random challenge. In their concrete construction, a challenge is a set of images, the secret user memorizes is a correspondence between images and numbers and the response is some basic function using addition of the digits (modulo 10). The authors provide experimental evidence that their scheme can be used by a human user. Namely, the secret can be memorized and the response can be computed within reasonable time by an average human user. The authors also propose a tool to help secret memorization. While the usability of their solution is not perfect, it is definitely a start and further research will hopefully yield protocols with better usability.

Security-wise, the authors prove that recovering the user's long-term secret from a number (below a certain bound) of random challenge-response pairs (non-adaptive queries) is equivalent to solving the random planted constraint satisfiability problem, and they state a conjecture about security of the latter. To support the conjecture, the authors prove the hardness of the problem for any *statistical* attacker, extending the results of [5]. Finally, it is proven that forging a response for a random challenge is equivalent to recovering the secret. The bound on the number of revealed challenge-response pairs corresponds to the maximum number of logins a user can execute, without endangering future sessions.

The construction and security results from [4] are very useful for our work, but we cannot use them as is. The problem is that it is not known whether security of their scheme holds when the attacker can see responses to maliciously chosen challenges (adaptive queries). We extend their analysis and prove a second conjecture that the unforgeability of their HC function family still holds if the adversary can make very few adaptive queries. Our Confirmed HAKE is designed to rely on such HC functions (whose security can tolerate very few adaptive queries): after the PAKE completion using the first response, the human user selects a random challenge and enters it into the terminal, who encrypts the challenge under the recently established session key and forwards the result to the server. The server decrypts, computes the response, and sends it, also encrypted to the terminal. The terminal decrypts and displays the response and the human user verifies it. If verification fails, the user needs to take measures against suspected terminal infection and possibly abort the long-term secret. Encrypting the terminal-server communication here is needed for authenticity in case of an honest terminal, to prevent a network adversary to ask the server maliciously chosen challenges. We show that this extended protocol limits the number of responses the attacker infecting the terminal can obtain for malicious challenges of its choice (in that case, the adversary will not be able to make the user pass the connection confirmation step). We argue that this addition, while adds a little bit more work for the human user, does not violate human computability for our instantiation, i.e., that the user can select a random challenge and verify the response. Furthermore, we show that the Confirmed HAKE provides explicit authentication assuming that the encryption scheme is secure authenticated encryption.

Our formal analysis on the HC function [4] demands a stronger conjecture stating *one-more* unforgeability. This is similar to the analysis of blind signatures that relies on the one-more unforgeability of RSA [6], but we consider a sequential version of the one-more security definition, that is weaker than the original one.

We want to note that, unlike [4], in our analysis, the bound on the number of challenge-response pairs the attacker can see does not correspond to the total number of logins, but only to the number of logins via compromised terminals, which is much more practical. This is because PAKE guarantees security against network attackers when end-points are secure: responses remain completely hidden to external players.

Unfortunately, it is not clear how to extend the results of [4] to expect resistance to many adaptive queries (so that we could have a simpler protocol without the confirmation step). The only possibility is the use of a pseudo-random function: after many adaptive queries, the response to a new challenge is still random-looking to any adversary. But for such functions, one needs additional help, hence our *device-assisted* scenario. One important advantage of such a stronger HC function family (tolerating many adaptive challenges, and thus also many non-adaptive challenges) is that responses are ephemeral secrets used once for authentication, but that can be revealed after use: as a consequence, a weaker variant of PAKE is enough, since resistance to off-line dictionary attacks is not required any more. We can expect more efficient constructions. Hence is our first construction in the device-assisted context. But to limit interactions with the device and avoid collisions on the inputs, we thereafter adopt a time-based challenge: $r = F_K(t)$, with an increasing counter t, based on an internal clock. While one cannot guarantee perfect synchronization between the device and the server, we can tolerate a slight time-shift since we

anyway use timeframes that are long enough for the human to enter the response read on the device (e.g., 30 seconds or 1 minute).

RELATED WORK. As we mentioned, there are numerous results about PAKE, from the seminal paper of Bellovin and Merritt [1], [2], but they offer no practical solutions for strong corruptions, in order to protect future sessions. There are various cryptographic schemes involving human participants, using graphical identification [7]–[14], some of them offer security against shoulder surfing. But they offer no security if the terminal is fully compromised.

Matsumoto and Imai [15] proposed the first scheme to deal with human identification through insecure channels (and via untrusted machines). The scheme has been improved by the follow up works [16]–[18]. However, the schemes are only secure given very few login sessions or require the human to memorize a long bitstring. We view the aforementioned paper by Blocki et al. [4] as an improvement over the results of this line of work.

Dziembowski [19] also considers the problem of human-based key-exchange, but in a setting where both parties are human and his scheme is only secure against a machine adversary assumed to be unable to solve CAPTCHAs.

There is a long sequence of papers [20]–[26] following the work by Hopper and Blum [27] offering protocols for the same problem of secure human identification over insecure channels, whose security is based on the Learning Parity with Noise problem. With error-correcting codes, such protocols could be adapted to generate deterministic responses (which is required by our HC function definition), but usability will not be good for the same reason as most HB-type protocols are not really suitable for humans. Personal devices generating one-time passwords have been commercially available for years [28], motivating IETF to standardize their constructions and use in many protocols [29]–[33]. The work [33] is particularly relevant as it defined a time-based one-time password algorithm based on HMAC [32]. Interestingly, while dedicated token generators are the most secure, software applications running on mobile phones are now commonly used [34].

Some papers also explore PAKE schemes with one-time passwords. Paterson and Stebila [35] define a security model for one-time PAKE, explicitly considering the compromise of past (and future) one-time passwords, but still recovering the security after a compromise, thanks to the ephemeral property of the one-time password and its change over the time. Unfortunately, their construction is a generic one, using a PAKE as a black-box. It thus cannot be more efficient than a PAKE, whereas preventing off-line dictionary attacks is not required in this setting. Our goal is to get a more efficient construction than any PAKE protocol, which we achieve with our device-assisted HAKE in Section VI. The authors of [35] mention the possibility of using a secure token to generate the one-time passwords and then running one-time-PAKE on it, but they did not provide an explicit protocol or security analysis.

OPEN PROBLEMS. We hope that our work will stimulate further results about secure human-compatible cryptographic function families. We leave to future works to formally prove the unforgeability property (against several adaptive queries) of the HC function from [4], and possibly finding other HC function families with such security. Those would allow to avoid additional devices and still have a completely proven efficient HAKE protocol. Improving the usability of the scheme from [4] will indeed imply improved HAKE protocols, and may have other applications. Another interesting question is to design a coin-flipping protocol with a human participant. Such protocol could be used within HAKE to prevent the attacker to ask malicious challenges. Eventually, after this first step of modeling HAKE protocols with symmetric long-term secrets shared between the user and the server, asymmetric secrets would be important to consider. This would be similar to the so-called *verifier-based PAKE* that helps moderate the impact of corruption of the server.

II. HUMAN AUTHENTICATED KEY EXCHANGE (HAKE)

A. HAKE Definitions

In this section, we define a human authenticated key exchange (HAKE) protocol, as an extension of [2].

PROTOCOL PARTICIPANTS. We fix the set of participants to be $\mathsf{ID} = \{U_\ell\}_\ell \cup \{T\} \cup \{S\}$, which contains finite number of human users U_ℓ, one terminal T and one server S. And we assume that each member is uniquely described by a bitstring. In the real life, each user U_ℓ can communicate with multiple servers via multiple terminals. But we justify below why considering a single terminal and a single server is sufficient.

HUMAN-COMPATIBLE COMMUNICATION. Here we present several notions that our protocol definition will use. Since it is hard to formalize human computational abilities, our definitions are not mathematically precise.
We say a message is *human-readable* if this is a short sequence of ASCII symbols, or images; *human-writable* if this is a short sequence of ASCII symbols[1]; *human-memorizable* if this is simple enough to be memorized by an average human, e.g., a simple arithmetic rule like "plus 3 modulo 10". A function is *human-computable* if an average human can evaluate it without help of additional resources other than his head, e.g., simple additions modulo 10. A set is *human-sampleable* if an average human can choose a message from the set at random according to the appropriate distribution without help of additional resources other than his head.

HAKE SYNTAX. We now formally describe a HAKE protocol.
Definition 1 (HAKE Protocol): A *human authenticated key exchange* protocol is an interactive protocol between a human user denoted $U \in \{U_\ell\}_\ell$ and the server S, via the terminal T. It consists of two algorithms:

[1]It is also possible to incorporate mouse clicks into that, but we do not deal with it for simplicity.

- A long-term key generation algorithm $\mathcal{LKG}$ which takes as input the security parameter and outputs a long-term key.
- An interactive key-exchange algorithm $\mathcal{KE}$ which is ran between U, T, and S. At the beginning, only U and S take as input the same long-term secret key and, at the end, T and S each outputs a session key sk_T and sk_S respectively. In case of additional explicit authentication, U and/or S may either *accept* or *reject* the connection.

The above algorithms must satisfy the following constraints:

- S can only communicate with T;
- U can only communicate with T, and
  - the message sent by T to U must be *human-readable*, and
  - the message sent by U to T must be *human-writable*;
- The long term secret and the state of U, if any, must be *human-memorizable* for the duration necessary.

The **correctness** condition requires that for every security parameter and for every long-term key output by $\mathcal{LKG}$, in any execution of $\mathcal{KE}$, U and S both accept the connection (in case of explicit authentication), T and S complete the protocol with the same session key ($\mathsf{sk}_T = \mathsf{sk}_S$).

B. Formal Security Model

In this section, we formally define the security model for a HAKE protocol, which is part of our main contributions. As already mentioned, the goal of a HAKE protocol is to ensure that a human user sharing the long-term secret with a server can help a terminal to establish a secure channel with the server, in presence of a very powerful attacker, including strong corruptions of terminals.

As usual, to model multiple and possibly concurrent (except for the human users) sessions we consider oracles π_P^j, where $j \in \mathbb{N}$ and $P \in \mathsf{ID}$. For human oracles, sessions can only be sequential, and not concurrent, meaning that humans are not allowed to run several sessions concurrently (a new session starts after the previous one ends). This is a reasonable assumption for human users. We note that since terminals do not store long-term secrets and do not preserve state between sessions, multiple terminal oracles model both multiple sessions ran from the same or different terminals.

Hence, in the following, we will consider several human users U_ℓ with different long-term secret keys, one terminal T, and one server S, with all the users' long-term secret keys. For all of them, multiple instances will model the multiple sessions (either sequential for U_ℓ, or possibly concurrent for T and S). However, while the server can concurrently run several sessions, we will also limit it to one session at a time with each user: the server will not start a new session with a user until it finishes the previous session with the same user.

Because of our specific context with a human user, there is a direct communication link between the user and the terminal, and so we can assume that the channels between instances $\pi_{U_\ell}^i$ and π_T^j are authenticated and even private (unless the terminal oracle is *compromised*, as defined below), whereas the communication between the terminal and the server is over the internet, and so the channels between instances π_T^j and π_S^k are neither authenticated nor private.

SECURITY EXPERIMENTS. We consider the following security experiments associated with a given HAKE protocol and an adversary $\mathcal{A}$, to define the two classical security notions for authenticated key exchange: privacy (or semantic security of the session key) and authentication. In these experiments, the adversary $\mathcal{A}$ can make the following queries:

- $\texttt{Compromise}(j, \ell)$, where $j, \ell \in \mathbb{N}$ – As the result of this query, the terminal-oracle π_T^j is considered to be *compromised*, and the adversary gets its internal state, i.e. the random tape, temporary variables, etc. If the terminal-oracle π_T^j is not linked yet to a user, it is linked to user U_ℓ with the user oracle $\pi_{U_\ell}^i$ for a new index i, otherwise ℓ is ignored;
- $\texttt{Infect}(j)$, where $j \in \mathbb{N}$ – As the result of this query, the terminal-oracle π_T^j is considered to be *infected*. WLOG, we limit this query to *compromised* terminals only;
- $\texttt{SendTerm}(j, M)$, where $j \in \mathbb{N}$ and $M \in \{0,1\}^* \cup \{\texttt{Start}(\ell)\}$ – This sends message M to π_T^j. A specific $\texttt{Start}(\ell)$ message asks the terminal to initiate a session, to be done with a user oracle $\pi_{U_\ell}^i$ for a new index i. But only if the terminal-oracle π_T^j is not linked yet to a user, otherwise ℓ is ignored. To compute its response to $\mathcal{A}$, π_T^j may internally talk to its linked human oracle according to the protocol. In addition, if π_T^j is *compromised*, it will additionally give to $\mathcal{A}$ the messages exchanged with its linked human oracle[2].
- $\texttt{SendServ}(k, M)$, where $k \in \mathbb{N}$ and $M \in \{0,1\}^*$ – This sends message M to oracle π_S^k. The oracle computes the response according to the corresponding algorithm and sends the reply to $\mathcal{A}$.
- $\texttt{SendHum}(j, M)$ where $j \in \mathbb{N}$ and $M \in \{0,1\}^*$ (and human-readable) – This sends a message to the π_T^j-linked human oracle $\pi_{U_\ell}^i$ on behalf of π_T^j. This is allowed only if the terminal π_T^j is *infected* (and thus *compromised*, which implies the existence of a partenered human oracle). The oracle computes the response according to the corresponding algorithm and sends the reply to $\mathcal{A}$.
- $\texttt{Test}(j, P)$, where $j \in \mathbb{N}$ and $P \in \{T\} \cup \{S\}$ – If sk_P has been output by π_P^j, then one looks at the internal bit b (flipped once for all at the beginning of the privacy experiment, while $b = 1$ in the authentication experiment). If $b = 1$, then $\mathcal{A}$ gets the real session key sk_P, otherwise it gets a uniformly random session key. This query is only allowed if π_P^j is fresh (defined below).

In the **privacy** experiment, after having adaptively asked several of these oracle queries, the adversary $\mathcal{A}$ outputs a bit b' (a guess on the bit b involved in the $\texttt{Test}$-queries). The intuition is that the adversary should not be able to distinguish

[2]The messages to the human oracle can be already known to the adversary as they are a function of the oracle's random tape. But we give the adversary the whole communication for convenience.

the real session keys from independent random strings. While in the **authentication** experiment, the goal of the adversary is to make an honest party to successfully complete the protocol execution thinking it "built a secure session" with the right party, whereas that is not the case. In order to formally define the goals and the advantages of the adversary, we present the notions of partnering and freshness, as well as the flags accept and terminate.

FLAGS. In order to model authentication, we follow BPR [2], who defined two flags: accept essentially means that a party has all the material to compute the session key while terminate means that a party thinks that it completes the protocol execution thinking it communicates with the expected other party (a human user in our case). These two flags are initially set to `False`, and they are explicitly set to `True` in the description of the protocol. Note that in Definition 1 U and/or S *accept* if and only if in the end the terminate flag is set to `True`, otherwise, U and/or S *reject*.

PARTNERING. Whereas $\pi_{U_\ell}^i$ and π_T^j are declared as *linked* at the initialization of the communication because of the authenticated channels between users and the terminal, partnering between $\pi_{U_\ell}^i$ and π_S^k is *a posteriori*: they are indeed declared *partners* in the end of the protocol execution if they use the same long-term key and both accept. Then we define partnering between π_T^j and π_S^k, by saying that they are declared *partners* if π_S^k and U_ℓ^i are partners and U_ℓ^i is linked to π_T^j.

FRESHNESS. Informally, the freshness denotes oracles that hold sessions keys that are not trivially known to the adversary. For $P \in \{T\} \cup \{S\}$, the oracle π_P^j is *fresh*, if no `Test`-query has been asked to π_P^j nor its partner, and none of π_P^j or its partner have been *compromised* (π_T^j is fresh if it has not been compromised, and π_S^k is fresh if the terminal *linked* to the partner human user has not been compromised.)

SECURITY NOTIONS. In the **privacy** security game, the goal of the adversary is to guess the bit b involved in the `Test`-queries. Then we measure the success of an adversary $\mathcal{A}$, that outputs a bit b', by $\mathsf{Adv}^{\mathsf{priv}}_{\mathsf{HAKE}}(\mathcal{A}) = 2 \cdot \Pr[b' = b] - 1$. This notion implies *implicit* authentication, which essentially means that no one else than the expected partners share the session key material.

For *explicit* authentication, we define the **authentication** security game: the goal of the adversary is essentially to make a player terminate (flag terminate set to true) without an accepting partner (flag accept set to true). But in our case with *compromised* or even *infected* terminals, this is a bit more complex than usual. We thus split the authentication security in two parts:

- Server-authentication: a user oracle should not successfully terminate a session if there is not exactly one partner server oracle that has accepted. Then, we denote $\mathsf{Adv}^{\mathsf{s\text{-}auth}}_{\mathsf{HAKE}}(\mathcal{A})$ the probability the adversary $\mathcal{A}$ makes such a bad event happens;
- User-authentication: a server oracle should not successfully terminate a session if there is not exactly one partner user oracle that has accepted. Then, we denote $\mathsf{Adv}^{\mathsf{u\text{-}auth}}_{\mathsf{HAKE}}(\mathcal{A})$ the probability the adversary $\mathcal{A}$ makes such a bad event happens.

Eventually, for any adversaries $\mathcal{A}, \mathcal{B}$ there exists an adversary $\mathcal{C}$ against the authentication security for which we define $\mathsf{Adv}^{\mathsf{auth}}_{\mathsf{HAKE}}(\mathcal{C}) = \max\{\mathsf{Adv}^{\mathsf{s\text{-}auth}}_{\mathsf{HAKE}}(\mathcal{A}), \mathsf{Adv}^{\mathsf{u\text{-}auth}}_{\mathsf{HAKE}}(\mathcal{B})\}$.

PASSIVE SESSIONS. We now define a new notion of *passive session*, which extends the `Execute`-queries in the standard BPR model [2]. Recall that `Execute`-queries allow the adversary to get full transcripts of communication between honest parties. Even though the same can be achieved via `Send`-queries, in the security analyses it is useful to count the number of observed honest sessions and the number of maliciously altered sessions separately. In addition, we will not limit to full sessions: the adversary can stop forwarding honest flows, making the session abort. Then, there can be *passive full/partial-sessions*:

Definition 2 (Passive Session): A (full or partial) session between oracles π_T^j and π_S^k is called *passive*, if the messages of all queries $\texttt{SendTerm}(j, \cdot)$ or $\texttt{SendServ}(k, \cdot)$ are either $\texttt{Start}(\cdot)$ or themselves an output of one of these two queries type. If flows are numbered, this also implies that the actual order of flows between T and S has not been modified. If all the outputs have been forwarded as inputs, this is a *passive full-session*, otherwise this is a *passive partial-session*.

Sessions that are not passive are called *active*, since the adversary altered something in the honest execution.

We believe this notion is stronger than the `Execute`-queries defined in the BPR security model, since the adversary does not need to decide from the beginning if all the exchanges will be passive or not. $\mathcal{A}$ can start with a passive sequence and decide at some point to stop (passive partial-session) or behave differently in an adaptive way (active session).

RESOURCES OF THE ADVERSARY. When doing security analyses, for every adversary and its privacy and authentication advantages, one also has to specify the adversarial resources such as the running time t, the number of oracle queries, the number of player instances, and the numbers $n_{\mathsf{passive}}/n_{\mathsf{active}}$ of (fully) passive and active sessions the adversary needs.

DISCUSSION. We discuss a bit more about our security definitions to explain why they capture the practical threats. First, a passive network adversary is able to observe legitimate communications via `SendServ` and `SendTerm`-queries (these will satisfy the passive sessions definition). An active network adversary can modify legitimate messages or impersonate a terminal or a server by injecting some messages of its choice, again, via `SendServ` and `SendTerm`-queries. This models, in the standard way, possible insecurity (in terms of privacy or authentication) of the network channel between terminals and servers.

Passive-insider attacks (such as keylogger and screen capture malware compromising computers or their browsers) are

modeled by Compromise-queries followed by SendTerm-queries. The former gives the adversary full information about the terminal's internal state, including its random coins and registers' contents, and the latter reveals to the adversary the inputs from the human.

We consider even more powerful attackers who can take full control of the computers or some of their crucial applications such as browsers. In this case, in addition to learning the internal state and all the inputs, the adversary can impersonate the honest terminal while sending adaptively selected messages to the human. We model this by Infect and SendHum-queries. Our model captures all the above scenarios and moreover, it takes into account the possibility of multiple simultaneous attacks, such as colluding network and malware adversaries. One can notice that attacks involving Infect-queries are stronger than those with Compromise-queries: when an adversary infects a terminal, it takes full control on it, with knowledge of its internal state, and thus plays on its behalf, using SendServ and SendHum-queries.

Note that in any case, we are concerned with the security of a new session, in terms of privacy and authentication, over an honest terminal, that is neither compromised nor infected. Such security should be guaranteed even though the other sessions involving the same human with the same long-term secret were carried over compromised terminals, and if possible even over infected terminals. We model privacy via the Test-query and with the appropriate privacy advantage definition. We model authentication via the corresponding advantage definition.

We also stress that we do not consider corruption of the long-term secrets, since they are known by the users and the server only, and we do not allow to corrupt them. Would the long-term secret be leaked, we cannot guarantee any security for future sessions. The interesting open problem of dealing with such corruptions could be addressed using an asymmetric long-term secret: a verifier-based variant that would just provide an encoded version of the user's secret to the server.

III. Building Blocks

For the sake of completeness, the building blocks that will be used in our constructions are detailed in the full version [36]. Since most of the details are useful for the proofs only, we just recall or present here the most important descriptions.

A. Human-Compatible Function Family

The protocols we propose in the next sections use special function families, which we call *human-compatible (HC)*.

Human-Compatible Function Family: Syntax. A human-compatible (HC) function family is specified by the challenge space $\mathcal{C}$, the key generation algorithm $\mathcal{KG}$, which takes input the security parameter and outputs a key K, and the *challenge-response* function F that takes a key K and a challenge $x \in \mathcal{C}$ and returns the response $r = F_K(x)$. We require that (see Section II-A for the definitions):

1) for every K output by $\mathcal{KG}$ and every $x \in \mathcal{C}$, both x and $F_K(x)$ are human-writable and human-readable;
2) $\mathcal{C}$ is human-sampleable.

We also define the Only-Human HC Function Family (where an additional device is excluded), which is the human-compatible function family that also has:

1) for every K output by $\mathcal{KG}$, $F_K(\cdot)$ is human-computable;
2) every K output by $\mathcal{KG}$ is human-memorizable;

Human-Compatible Function Family: Security. In an authentication protocol with challenge-response pairs, intuitively, we would like that any successful authentication to a server should involve an evaluation of the function by the human user. So we expect no compromised/infected terminal to successfully authenticate to the server one more time than it interacted with the human. The security notion from the function is thus a kind of one-more unforgeability [6]. But here, any query to an $F_K(\cdot)$-oracle should help to immediately answer $F_K(x)$ to the current challenge x, since a second challenge will come from a new session that has closed the previous one, and so the previous challenge is obsolete: the adversary cannot store the $n+1$ challenges, ask n queries, and answer the $n+1$ initial challenges. In our protocols, the adversary gets a random challenge (GetRandChal-query), can ask any $F_K(\cdot)$-query (GetResp-query), but should answer that challenge (TestResp-query), otherwise the failure is detected. After too many failures (recorded in the unvalidated-query counter ctr) one may restrict oracle queries. Hence our following security notion which formalizes these restrictions to the adversary.

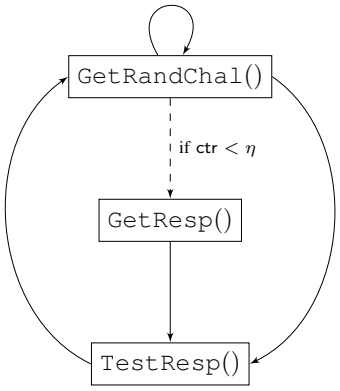

Fig. 1. Graph of the sequential oracle calls in the η-unforgeability experiment

η-Unforgeability. As said above, we thus define a kind of sequential one-more unforgeability experiment, with a limit η on the unvalidated-query counter ctr, where the queries follow the graph presented on Figure 1. Given an HC function family F, an adversary $\mathcal{A}$, and a public parameter η, one first generates K with $\mathcal{KG}$ and initializes ctr $\leftarrow 0$. Then the adversary can ask the following queries, with possible short loops on the GetRandChal-query and direct TestResp-attempt right after getting the challenge:

1) GetRandChal() – It picks a new $x \xleftarrow{\$} \mathcal{C}$, marks it *fresh* and outputs it;
2) GetResp(x^*) – If $\mathsf{ctr} < \eta$ and $x^* \in \mathcal{C}$, it returns $F_K(x^*)$ and increments ctr. It also marks the *fresh* x as *unfresh*. Otherwise, it outputs $\perp$;
3) TestResp(r) –
   - If $F_K(x) = r$ and x is *fresh*, the adversary *wins*;
   - If $F_K(x) = r$ and x is *unfresh*, it decrements ctr, marks x as *used*, and outputs 1;
   - Otherwise, it outputs 0.

Because of the sequential iterations, any TestResp-query relates to the previous GetRandChal-query. One can thus consider one memory-slot to store the challenges, but one only at a time: any new challenge replaces the previous one. The dashed line from GetRandChal to GetResp emphases the restriction on the number of unvalidated queries. When $\mathsf{ctr} \geq \eta$, the adversary has no more choice than immediately trying an answer for the random challenges. The bound η represents the maximum gap that is allowed at any time between the number of GetResp-queries and the number of correct TestResp-queries. Note that a random challenge x can only be either *fresh*, *unfresh*, or *used*, and that marking it as one of those erases the other flags. Intuitively, a *fresh* challenge has not been compromised in any way, and succeeding at a TestResp on it would indicate the unforgeability has been breached, hence the winning status for the adversary, and the experiment stops. A challenge can switch to the *unfresh* state if the adversary asks the GetResp-oracle for an answer. There are only two ways for the experiment to stop: if the adversary wins with a correct TestResp-query on a fresh challenge; or if the adversary aborts, it then looses the game. We stress that the adversary can query the GetResp-oracle on any x^* of its choice, and so possibly different from the current challenge x obtained with the previous GetRandChal-query. But we give it a chance to still answer correctly to the challenge x with the correct TestResp-query that, on an *unfresh* challenge, cancels the increment of the counter ctr. This counter represents the gap between the number of GetResp-queries and the number of correct TestResp-queries on random challenges. When one limits ctr to be at most 1, any GetResp-query should be immediately followed by a correct TestResp-query (one-more unforgeability).

This definition is a weaker notion than the one-more unforgeability [6], but still allows the adversary to exploit malleability: For example, with the RSA function, for a random challenge y, the adversary can ask a GetResp-query on any $y' = y \cdot r^e \bmod n$, for an r of its choice, so that it can then extract an e-th root of y. But this would not help it to answer a next fresh challenge.

2-PARTY η-UNFORGEABILITY. Unfortunately, the above clean security notion is not enough for our applications, as client-server situations and man-in-the-middle attacks allow more complex ordering of the queries by the adversary. We therefore present a variant of this experiment below, that is suitable for a protocol involving two parties (hence in the following $b \in \{0, 1\}$).

Given an HC function family F, an adversary $\mathcal{A}$, and a public parameter η, one first generates K with $\mathcal{KG}$ and initializes $\mathsf{ctr} \leftarrow 0$. Then the adversary can ask the following queries:

1) GetRandChal(b) – It picks a new $x_b \xleftarrow{\$} \mathcal{C}$, marks it *fresh* and outputs it;
2) GetResp(x^*) – If $\mathsf{ctr} < \eta$ and $x^* \in \mathcal{C}$, it returns $F_K(x^*)$ and increments ctr. It also marks all *fresh* x_b as *unfresh*. Otherwise, it outputs $\perp$;
3) TestResp(r, b) – If x_b exists:
   - If $F_K(x_b) = r$ and x_b is *fresh*, the adversary *wins*;
   - If $F_K(x_b) = r$ and x_b is *unfresh*, it decrements ctr, marks x_b as *used* and outputs 1;
   - Otherwise, it outputs 0.

The main difference with the previous experiment are the two memory-slots for challenges x_0 and x_1. But still, any GetResp-query must be followed by a correct TestResp-query to limit ctr from increasing too much.

The advantage of any adversary $\mathcal{A}$ against the unforgeability, $\mathsf{Adv}_F^{\eta\text{-uf}}(\mathcal{A})$ is the probability of winning in the above experiment (with a correct TestResp-query on a *fresh* challenge). Such a success indeed means that the adversary found the response for a new random challenge, without having asked for any GetResp-query.

The resources of the adversary are the polynomial running time and the numbers $q_\mathsf{c}, q_\mathsf{r}, q_\mathsf{t}$ of queries to GetRandChal, GetResp and TestResp oracles, respectively. Of course it is crucial whether there are secure instantiations of HC function families. We propose some in Section V.

INDISTINGUISHABILITY. For some constructions, we will expect the sequence of answers $\{F_K(x_i), i = 0, \ldots, T\}$ for challenges x_i (either adversarially chosen or not) to look random, or at least any new element in the sequence is not easy to predict from the previous ones.

For the sake of simplicity, we assume that there exists a global distribution $\mathcal{D}$ with large enough entropy D such that any such sequence is computationally indistinguishable from $\mathcal{D}^{T+1}$: We denote $\mathsf{Adv}_F^{\text{dist-}c}(\mathcal{D}, \mathcal{A})$ the advantage the adversary $\mathcal{A}$ can get in distinguishing the sequence $\{y_0 = F_K(x_0), \ldots, y_{c-1} = F_K(x_{c-1})\}$ for a random K, from $(y_0, \ldots, y_{c-1}) \xleftarrow{\$} \mathcal{D} \times \ldots \times \mathcal{D}$. For the latter distribution, the probability to guess y_{c-1} from the view of $(y_0, \ldots, y_{c-2})$ is $1/2^D$.

(Weakly) pseudo-random functions definitely satisfy this property. But from a more practical point of view, the function implemented in the RSA SecurID device [28] is believed to satisfy it too, with the x_i being a time-based counter.

B. Commitment Scheme

We will also use a commitment scheme, a primitive allowing a user to commit on a value x so that the receiver does not learn any information about x, but with the guarantee that the user will not be able to change his mind later.

COMMITMENT SCHEME: SYNTAX AND SECURITY. A (non-interactive) commitment scheme $\mathcal{CS}$ is defined by Setup that defines the global public parameters, and two other algorithms:

- $\mathsf{Com}(x)$: on input a message x, and some internal random coins, it outputs a commitment c together with an opening value s;
- $\mathsf{Open}(c, s)$: on input a commitment c and then opening value s, it outputs either the committed value x or $\perp$ in case of invalid opening value.

The **correctness** condition requires that for every x, if $(c, s) = \mathsf{Com}(x)$, then $\mathsf{Open}(c, s)$ outputs x. The usual **security notions** for commitment schemes are the *hiding* property, which says that x is hidden from c, and the *binding* property, which says that once c has been sent, no adversary can open it in more than one way.

For the security of our protocols we will need additional properties, such as *extractability* (a simulator can extract the value x to which c will be later opened) and *equivocality* (a simulator can generate some fake commitments c it can later open to any x). These features are provided from trapdoors, generated by an alternative setup algorithm and privately given to the simulator. More details can be found in the full version [36]. But in the following, we will denote $\mathsf{Adv}_{\mathcal{CS}}(\mathcal{A})$ the advantage an adversary can get against any of these security notions.

COMMITMENT SCHEME: INSTANTIATION. An efficient instantiation, with all our expected security properties, in the random oracle model [37], can be described as follows:
Given a hash function $\mathcal{H}$ onto $\{0,1\}^\lambda$,

- $\mathsf{Com}(x)$: Generate $r \xleftarrow{\$} \{0,1\}^{2\lambda}$ and output $(c \leftarrow \mathcal{H}(x, r), s \leftarrow (x, r))$;
- $\mathsf{Open}(c, s = (x, r))$: if $\mathcal{H}(s) = c$, return x, otherwise, return $\perp$.

In the random oracle model, this simple scheme is trivially computationally *binding* ($\mathcal{H}$ is collision-resistant) and statistically *hiding* (for a large $r \in \{0,1\}^{2\lambda}$, there are almost the same number of possible r —actually, 2^λ— for any x, that would lead to the commitment c) in the ROM.

C. Password-Authenticated Key Exchange

We will also make (black-box) use of a password-authenticated key exchange (PAKE) protocol. A PAKE protocol is an interactive protocol between two parties who share a common low entropy secret (a password) for an execution with session id PAKEsid. At the end of the protocol, the parties output a session key. The correctness requires that any honest PAKE execution with matching passwords results in the parties outputting the same session key.

The security model can be defined in the UC framework [38], with an ideal functionality $\mathcal{F}_{\mathsf{pake}}$. We will denote $\mathsf{Adv}^{\mathsf{pake}}_{\mathsf{PAKE}}(\mathcal{S}, \mathcal{A}, \mathcal{Z})$ the advantage the distinguisher $\mathcal{Z}$ can get in distinguishing the ideal world with the simulator $\mathcal{S}$ and the real world with the adversary $\mathcal{A}$. Again, more details can be found in the full version [36], but an efficient instantiation, satisfying all our expected security requirements is the classical EKE [1] protocol that encrypts a Diffie-Hellman key exchange, using the password as encryption key. It has been proven UC-secure [39], under the Computational Diffie-Hellman assumption in the ideal-cipher model.

D. Authenticated Encryption

Eventually, for *explicit* authentication of the players, we will make use of an authenticated encryption scheme [40] $\mathcal{ES} = (\mathsf{Enc}, \mathsf{Dec})$, where decryption should fail when the ciphertext has not been properly generated under the appropriate key. This will thus provide a kind of key confirmation, as usually done to achieve explicit authentication. However, some critical data will have to be sent, hence a simple MAC would not be enough, privacy of the content is important too.

For an authenticated encryption scheme, there are two main security notions: The *semantic security*, a.k.a IND-CPA, prevents any information being leaked about the plaintexts, while the *integrity of ciphertexts*, a.k.a. INT-CTXT, essentially says that no valid ciphertext can be produced without the key. The definitions of the corresponding advantages $\mathsf{Adv}^{\mathsf{int\text{-}ctxt}}_{\mathcal{ES}}(\mathcal{A})$ and $\mathsf{Adv}^{\mathsf{ind\text{-}cpa}}_{\mathcal{ES}}(\mathcal{B})$, for any adversaries $\mathcal{A}, \mathcal{B}$ can be found in [40]. In addition, for adversaries $\mathcal{A}, \mathcal{B}$ there exists an adversary $\mathcal{C}$ for which we define $\mathsf{Adv}^{\mathsf{authenc}}_{\mathcal{ES}}(\mathcal{C}) = \max\{\mathsf{Adv}^{\mathsf{int\text{-}ctxt}}_{\mathcal{ES}}(\mathcal{A}), \mathsf{Adv}^{\mathsf{ind\text{-}cpa}}_{\mathcal{ES}}(\mathcal{B})\}$.

One simple way to achieve secure authenticated encryption is by using a generic Encrypt-then-MAC approach [40] or by using a dedicated scheme such as OCB [41].

IV. GENERIC HAKE PROTOCOLS

In this section, we propose two generic HAKE protocols. They build on a simple idea of composing a human-compatible (HC) function family with a password authenticated key exchange (PAKE) protocol. More precisely, a server chooses a random challenge x, the user U_ℓ's response is $r = F_{K_\ell}(x)$, where F is an HC function family and K_ℓ is the long-term secret shared between the user and the server. And finally the terminal and the server execute the PAKE on the one-time password r, as in [35]. As already mentioned, whereas the server supports concurrent sessions, since the human does not, there is no sense in maintaining multiple session states for one human user.

However, a straightforward replay attack is possible. The adversary can first just eavesdrop a session by compromising a terminal, and then play on behalf of the server with the observed challenge-response pair (x, r), even when the user uses an honest terminal. The main issue is that there is no reason for the challenge to be distinct in the various sessions if we do not add a mechanism to enforce it. In [35]'s constructions, they assume the server is stateful to prevent it. However, we can do better.

This is the goal of our first protocol: it adds a coin-flipping protocol between the terminal and the server to avoid either party to influence the challenge x, and thus to avoid the aforementioned replay attacks. We prove it secure (in terms of privacy, which implies implicit authentication) assuming security of commitments (underlying the coin-flipping), HC function family, and PAKE. However, the concrete security

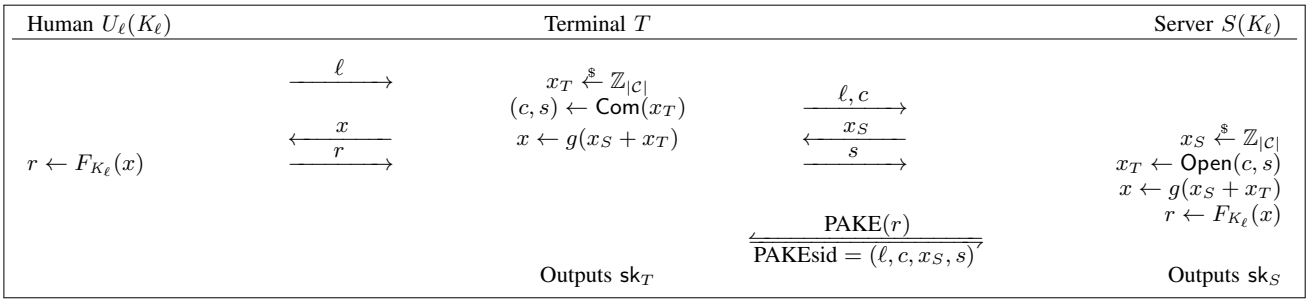

Fig. 2. Basic Generic HAKE Construction

depends on the bound η, which is large enough for our device-based HC function family, but the Only-Human HC function family construction we will propose in Section V (and its underlying hardness problem) does not tolerate a high η.

Hence, the goal of our second HAKE protocol is to add explicit authentication, which will help limiting the number of malicious challenge-response pairs the adversary can see, or at least to detect them: the user can then suspect the terminal to be infected. We still need the concrete HC function to tolerate at least one malicious challenge, but this remains a reasonable assumption.

A. The Basic Generic HAKE

Our first construction makes use of a commitment scheme, an HC function family, and a PAKE.

DESCRIPTION. Let $(\mathcal{KG}, F)$ be a human-compatible function family with challenge space $\mathcal{C}$, let $\mathcal{CS} = (\mathsf{Setup}, \mathsf{Com}, \mathsf{Open})$ be a commitment scheme, let PAKE be a password authenticated key exchange protocol and let $g : \mathbb{Z}_{|\mathcal{C}|} \to \mathcal{C}$ be a bijection. We construct the Basic HAKE $(\mathcal{LKG} = \mathcal{KG}, \mathcal{KE})$. Its interactive $\mathcal{KE}$ protocol is described on Fig. 2, here are the descriptions.

- $\mathcal{KE}$ execution:
  1) When the user invokes a terminal to establish a connection with the server, the terminal chooses its part of the challenge x_T, and commits it for the server. It also sends the user's identifier ℓ;
  2) Upon receiving the commitment, the server waits until any previous session for U_ℓ finishes, then it chooses its part of the challenge x_S, and sends it in clear to the terminal;
  3) The terminal then combines both parts x_T and x_S to generate the challenge $x = g(x_S + x_T)$, and asks x to the user;
  4) Upon reading the challenge x, the user computes and writes down the response r for the terminal;
  5) When the terminal receives the response r from the human user, it opens its commitment to the server, and can already starts with the PAKE protocol execution;
  6) Upon receiving the opening value of the commitment, the server opens the latter to get x_T. It can then combine both parts x_T and x_S to generate the challenge $x = g(x_S + x_T)$, and compute the response r. It can then proceed with the PAKE protocol too.

  The terminal and the server both run the PAKE protocol with their (expected) common input r and session id PAKEsid that is the concatenation of the transcript. At the end of the PAKE execution, they come up with two session keys, sk_T and sk_S, respectively, that will be equal if both parties used the same r in the PAKE. Since we do not consider explicit authentication, accept and terminate flags are not set.

Correctness of the HAKE construction follows from correctness of the building blocks.

SECURITY ANALYSIS. For Basic HAKE, we only assess privacy of the session key, since this protocol does not provide explicit authentication.

Theorem 1: Consider the Basic HAKE protocol defined in Fig. 2. Let $\mathcal{A}$ be an adversary against the privacy security game with static compromises, running within a time bound t and using less than n_{comp} *compromised* terminal sessions, n_{uncomp} *uncompromised* terminal sessions, n_{serv} server sessions and $n_{\mathsf{active}} \leq n_{\mathsf{comp}} + n_{\mathsf{uncomp}} + n_{\mathsf{serv}}$ active sessions. Then there exist an adversary $\mathcal{B}_1$ attacking the 2-party n_{comp}-unforgeability of the HC function family with $q_{\mathsf{r}}, q_{\mathsf{c}}, q_{\mathsf{t}}$ queries of the corresponding type, an adversary $\mathcal{B}_2$ and a distinguisher $\mathcal{B}_3$ attacking UC-security of the PAKE with a simulator $\mathcal{S}_{\mathsf{pake}}$ as well as an adversary $\mathcal{B}_4$ against the commitment scheme, all running in time t, such that

$$\mathsf{Adv}^{\mathsf{priv}}_{\mathsf{HAKE}}(\mathcal{A}) \leq \mathsf{Adv}^{n_{\mathsf{comp}}\text{-}\mathsf{uf}}_{F}(\mathcal{B}_1) + 2 \cdot \mathsf{Adv}^{\mathsf{pake}}_{\mathsf{PAKE}}(\mathcal{S}_{\mathsf{pake}}, \mathcal{B}_2, \mathcal{B}_3) + 6 \cdot \mathsf{Adv}_{\mathcal{CS}}(\mathcal{B}_4)\ ,$$

where $q_{\mathsf{r}} \leq n_{\mathsf{comp}}$, $q_{\mathsf{t}} \leq n_{\mathsf{active}}$, and $q_{\mathsf{c}} \leq n_{\mathsf{uncomp}} + n_{\mathsf{comp}} + n_{\mathsf{serv}}$.

The proof is in the full version [36].

DISCUSSION. Concrete security of the HC function family is definitely the most crucial compared to that of other building blocks, since it is hard to balance strong security and usability. This is why we emphasize this in the above theorem.

We note that the sessions which lead to `TestResp`-queries have *non-oracle-generated* flows and therefore correspond to classical on-line dictionary attacks: the adversary simply tries to impersonate the user/terminal to the server (or vice-versa), with a guess for the answer r (unless the query has

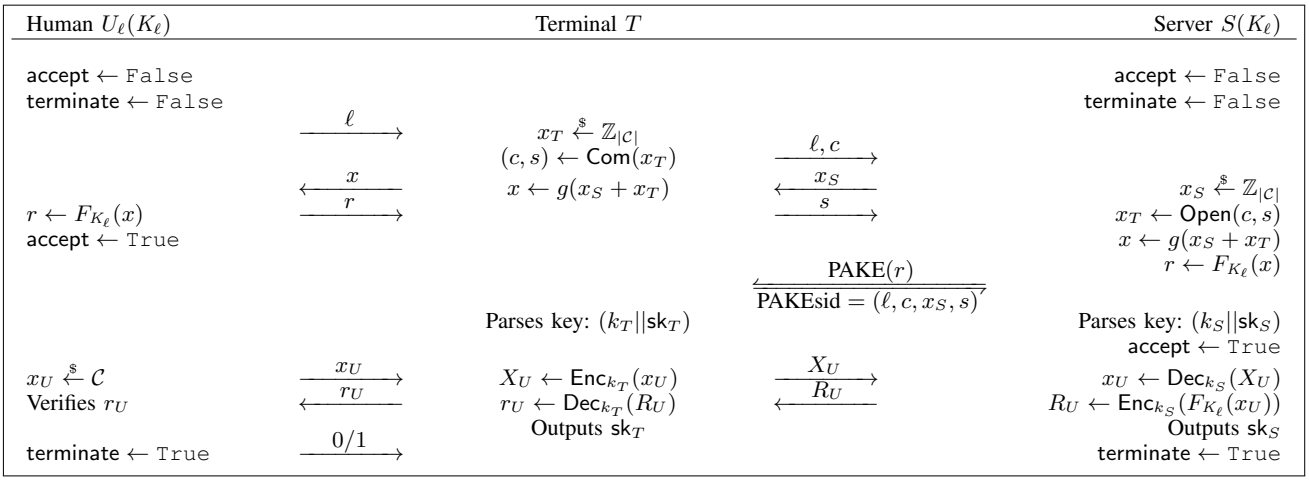

Fig. 3. Confirmed HAKE Construction

been asked to the GetResp-oracle). Indeed, sessions with GetResp-queries on the exact challenges have *compromised* terminals and correspond to a spyware keylogger that records random challenge-response pairs. If using such a compromised terminal can be considered rare, this remains reasonable. Eventually, the sessions which lead to GetResp-queries on different challenges are the most critical, but they should likely fail. And we expect them to be quite exceptional. As remarked above, such sessions likely conclude to a failure: no concrete session is established with the server the user wanted to connect to (if the HC function family is still secure after such adaptive queries). If the user can detect such a failure, he can run away from this terminal. We now propose a way for the user to detect such a dangerous terminal, and thereafter take appropriate measures. At the same time, our next proposal will achieve *explicit* authentication.

B. The Confirmed HAKE

We now enhance the Basic Generic HAKE by adding two confirmation flows (see Fig. 3) that allow the user to detect a bad behavior of the adversary, who compromised the device, and take appropriate measures. This can happen in two different scenarios: the adversary just compromised a terminal and additionally plays on behalf of the server, which allows it to ask any query to the user through the terminal, or the adversary infected a terminal that allows it to directly ask any query to the user.

As said above, such dangerous cases lead to no connection with the expected server. The user will thus check whether he built a secure session with the expected server, who should be able to answer to a fresh random challenge. This is performed under the fresh key, established with the PAKE, using a secure authenticated encryption. As shown below, the two additional flows will not only provide *explicit* authentication, but also allow the user to detect such bad events and take measures. For this, it is important that the user does not start multiple sessions concurrently, which is anyway not realistic for a human (as already noticed above).

DESCRIPTION. The protocol is similar to Basic HAKE, but it uses an additional building block, an authenticated encryption scheme $\mathcal{ES} = (\mathsf{Enc}, \mathsf{Dec})$, that is used in the new last stage of the protocol. The description is in Fig. 3.

Since we now consider the authentication of the players, we additionally include accept and terminate flags in the protocol: The user U_ℓ accepts after sending the first response while the server S accepts after the PAKE. Then both terminate when they have the confirmation of the other partner. More precisely, the user terminates after sending the last bit (1 for acceptance and 0 for rejection) to the terminal (thus having verified the server's response in the last stage), and the server terminates after sending the encrypted response (thus having checked the terminal can generate a valid ciphertext).

Note that if the protocol terminates, sk_T and sk_S must be equal, since our additional flows act as confirmation flows for the PAKE.

SECURITY ANALYSIS. We now present Theorem 2 regarding the security of our Confirmed HAKE in the HAKE privacy and authenticity experiment.

While it relies on the same security properties of PAKE, authenticated encryption, commitment scheme and HC function, a critical parameter is added, the number of human sessions that *reject* in the end.

Indeed, the explicit authentication property we achieve means that any attempt at issuing an adaptive query unrelated to the challenge will likely lead to a failure of the PAKE protocol, that can in turn be detected by the human, as he doesn't get the answer to x_U he looked for. This allows to use a much stricted η in the HC unforgeability game (even $\eta = 1$ for a very strict human user), which is a much more reasonable goal for an HC function family such as the one derived from [4], that we will present in Section V-B

Theorem 2: Consider the Confirmed HAKE protocol defined in Fig. 3. Let $\mathcal{A}, \mathcal{A}'$ be adversaries against the privacy and authenticity security game of HAKE within a time bound

t and using less than n_{comp} *compromised* terminal sessions, n_{uncomp} *uncompromised* terminal sessions, n_{serv} server sessions, $n_{\mathsf{active}} \leq n_{\mathsf{comp}} + n_{\mathsf{uncomp}} + n_{\mathsf{serv}}$ active sessions and n_{hr} human session that *reject* in the end. Then there exist two adversaries $\mathcal{B}_1, \mathcal{B}'_1$ attacking the 2-party $(n_{\mathsf{hr}}+1)$-unforgeability of HC function family with $q_{\mathsf{r}}, q_{\mathsf{c}}, q_{\mathsf{t}}$ queries of the corresponding type, two adversaries $\mathcal{B}_2, \mathcal{B}'_2$ and two distinguishers $\mathcal{B}_3, \mathcal{B}'_3$ attacking UC-security of the PAKE with the simulator $\mathcal{S}_{\mathsf{pake}}$, two adversaries $\mathcal{B}_4, \mathcal{B}'_4$ against the authenticated encryption, as well as two adversaries $\mathcal{B}_5, \mathcal{B}'_5$ against the commitment scheme, all running in time t, such that

$$\begin{aligned}\mathsf{Adv}^{\mathsf{priv}}_{\mathsf{HAKE}}(\mathcal{A}) &\leq \mathsf{Adv}_F^{(n_{\mathsf{hr}}+1)\text{-}\mathsf{uf}}(\mathcal{B}_1) + 2\cdot\mathsf{Adv}^{\mathsf{authenc}}_{\mathcal{ES}}(\mathcal{B}_4)\\ &\quad + 6\cdot\mathsf{Adv}_{\mathcal{CS}}(\mathcal{B}_5) + 2\cdot\mathsf{Adv}^{\mathsf{pake}}_{\mathsf{PAKE}}(\mathcal{S}_{\mathsf{pake}},\mathcal{B}_2,\mathcal{B}_3),\\ \mathsf{Adv}^{\mathsf{auth}}_{\mathsf{HAKE}}(\mathcal{A}') &\leq \mathsf{Adv}_F^{(n_{\mathsf{hr}}+1)\text{-}\mathsf{uf}}(\mathcal{B}'_1) + 2\cdot\mathsf{Adv}^{\mathsf{authenc}}_{\mathcal{ES}}(\mathcal{B}'_4)\\ &\quad + 6\cdot\mathsf{Adv}_{\mathcal{CS}}(\mathcal{B}'_5) + 2\cdot\mathsf{Adv}^{\mathsf{pake}}_{\mathsf{PAKE}}(\mathcal{S}_{\mathsf{pake}},\mathcal{B}'_2,\mathcal{B}'_3),\end{aligned}$$

where $q_{\mathsf{r}} \leq 2n_{\mathsf{comp}}$, $q_{\mathsf{t}} \leq n_{\mathsf{active}}$, $q_{\mathsf{c}} \leq n_{\mathsf{comp}} + n_{\mathsf{uncomp}} + n_{\mathsf{serv}}$. The proof is provided in the full version [36].

Remark 1: We also note that given the confirmation phase, and assuming the strong policy of resetting all credentials if the confirmation phase fails, the coin-flipping part is no longer necessary for the security proof: we could let the server choose the challenge during the first phase and the human in the second one (to avoid one player being to make replay attacks). We chose to keep it as part of the protocol because, first, this would not reduce the number of flows since the terminal always initiates such a connection, and second, without coin-flipping a network attacker could test adaptive challenges. The confirmation phase would fail, but there is no real need for the user to take severe measures and change the long-term secret in such a weak attack. Hence we prevent adaptive tests (from network attacks) with coin-flipping, which may be useful if a policy a little weaker is in use, such as resetting only if there is suspicion of terminal infection.

V. Human-Compatible Function Family Instantiation

In Section III, we proposed instantiations for the HAKE building blocks, except for the HC function family. We now focus on the latter in this section.

A. Token-Based HC Function Family Instantiation

First, we introduce a simple token-based HC function family. This assumes that the human is in possession of a simple device on which it can input challenge x and get the response $r \leftarrow F_K(x)$. The device will store K and perform the computation, but the human is still responsible for the communication with the terminal.

This allows us to use strong cryptographic primitives. For instance, we could set $K \xleftarrow{\$} \{0,1\}^\lambda$ and $F_K : [\![0,9]\!]^{t'} \to [\![0,9]\!]^t$ a pseudorandom function. In the random oracle model (for modeling $\mathcal{H}$ in $F_K(x) = \mathcal{H}(K\|x)$), we have $\mathsf{Adv}_F^{\eta\text{-}\mathsf{uf}}(\mathcal{A}) \leq 10^{-t}$ for any adversary and any η, since an adversary can just guess by chance the answer to a fresh challenge. Note that this function is obviously *human-readable*, *human-writable* and *human-sampleable* as its input/output are numbers in basis 10 so it is an HC function family.

Hence this function family is a good candidate to use in our Basic generic HAKE protocol from Section IV-A or its simplified version from Section VI-A.

B. Only-Human HC Function Family Instantiation

However, avoiding such devices would be much better in practice. We are thus interested in the Only-Human HC function family that would not require anything beyond simple human memory and brain computation power. Since such a function is necessarily weaker, we will use it in our confirmed HAKE protocol from Section IV-B, that has a much tighter control over adaptive queries and therefore requires weaker security properties from the HC function family.

1) Construction: We present a candidate based on the construction of Blocki et al [4], which security is based on [5]: Consider a challenge space $\mathcal{C} = \mathcal{X}_l^t \subseteq [n]^{lt}$, where $[n] = \{1,\ldots,n\}$ is the set of n integers each representing one of the n variables and $\mathcal{X}_l$ denotes the space of ordered clauses of l variables without repetition. The parameter t indicates that each challenge consists of t independent clauses, i.e., "small" challenges. The key generation algorithm $\mathcal{KG}$ of our HC function family takes as input a parameter n, then outputs a random mapping $\sigma : [n] \to \mathbb{Z}_d$ as the key K, where the integer d is a constant. Usually we set $d = 10$ because most humans are familiar with computations on digits. Let $\sigma^l : [n]^l \to \mathbb{Z}_d^l = (\sigma,\cdots,\sigma)$ denote the mapping that applies σ to each element of an l-tuple. Using a public human-computable function $f : \mathbb{Z}_d^l \to \mathbb{Z}_d$ that is instantiated later, the challenge-response function F takes a key $K = \sigma$ and a challenge $x \in \mathcal{C}$ as inputs, and returns a response $r = F_K(x)$. Here $F_K : \mathcal{C} \to \mathbb{Z}_d^t$ is defined as a t-tuple $(t \geq 1)$ $(f\circ\sigma^l,\cdots,f\circ\sigma^l)$, where $\circ$ indicates the function composition. For instance, if $n = 100$, $l = 3$, $d = 10$, $t = 2$, $x = ((1,4,20),(3,36,41))$, $\sigma(i) = (i+3) \mod 10$ and $f = (x_1 - x_2 + x_3) \mod 10$, then $\sigma((1,4,20)) = (4,7,3)$, $\sigma((3,36,41)) = (6,9,4)$ and $F_K(x) = (0,1)$.

Given integers $k_1, k_2 > 0$, the function f is instantiated as $f_{k_1,k_2} : \mathbb{Z}_{10}^{10+k_1+k_2} \to \mathbb{Z}_{10}$, which is defined as follows:

$$f_{k_1,k_2}(x_0,\ldots,x_{9+k_1+k_2}) = x_{\left(\sum_{i=10}^{9+k_1} x_i \mod 10\right)} + \sum_{i=10+k_1}^{9+k_1+k_2} x_i \mod 10.$$

Note that when f is instantiated as f_{k_1,k_2}, we have $l = 10 + k_1 + k_2$ and $d = 10$.

It is easy to see that such a function family is an Only-Human HC function family apart from the human memorization property. However, we can allow for images to represent the variables. As illustrated in [4], by using mnemonic helpers, humans are able to remember such mappings from images to digits. As an evidence, the primary author of [4] was able to memorize a mapping from $n = 100$ images to digits in 2 hours.

2) Security: In [4], the authors proved the intractability to answer to a new random challenge for the above HC function family instantiation based on the conjecture about the hardness of *random planted constraint satisfiability problems (RP-CSP)*. We briefly recall a special case of the RP-CSP conjecture, which we call the RP-CSP* conjecture, and its implied security theorem, both with our notations. For an in-depth review of those notions, the reader should refer to [4].

THE RP-CSP* CONJECTURE. Before stating this conjecture, we introduce some notations as in [4]. Denote $H(\alpha_1, \alpha_2) = |\{i \in [n] \mid \alpha_1[i] \neq \alpha_2[i]\}|$ as the Hamming distance between two strings $\alpha_1, \alpha_2 \in \mathbb{Z}_d^n$. Use $H(\alpha) = H(\alpha, \vec{0})$ to denote the Hamming weight of α. Then we say two mappings $\sigma_1, \sigma_2 \in \mathbb{Z}_d^n$ are *ε-correlated* if $H(\sigma_1, \sigma_2)/n \leq (d-1)/d - \varepsilon$.

Conjecture 1 (RP-CSP):* Consider the function f_{k_1,k_2} described above, for any $\varepsilon, \varepsilon' > 0$ and any probabilistic polynomial time (in n) adversary $\mathcal{A}$, there exists an interger $N \in \mathbb{N}$, such that for all $n > N$, $m \leq n^{\min\{(k_2+1)/2, k_1+1-\varepsilon'\}}$, we have $\mathsf{Adv}^{\mathsf{rand}}_{f_{k_1,k_2}}(\mathcal{A}, \varepsilon) = \mathsf{negl}(n)$, where $\mathsf{Adv}^{\mathsf{rand}}_{f_{k_1,k_2}}(\mathcal{A}, \varepsilon)$ is the probability that $\mathcal{A}$ outputs a mapping σ' that is ε-correlated with the secret mapping σ given m random "small" challenge-response pairs $\{(C_i, f_{k_1,k_2}(\sigma^l(C_i)))\}_{1 \leq i \leq m}$.

Remark 2: The RP-CSP conjecture in [4] is a general version of the RP-CSP* Conjecture 1, where f can be instantiated as other functions. Here, for simplicity, we only state the conjecture where $f = f_{k_1,k_2}$. In [4], the authors also prove strong evidence in support of the RP-CSP conjecture: it holds for any statistical adversary and any Gaussian Elimination adversary. As observed in [42], most natural algorithmic techniques have statistical analogues except the Gaussian Elimination.

BASIC η-UNFORGEABILITY. To state the security theorem in [4], we need the following "basic" HC security notion that is a "non-malleable" version of the η-unforgeability. It indeed assumes that asking a `GetResp`-query with an input different from the current random challenge should not help to answer this challenge correctly to the `TestResp`-query. Given an HC function family F, an adversary $\mathcal{A}$, and a public parameter η, one first generates K with $\mathcal{KG}$ and initializes $\mathsf{ctr} \leftarrow 0$. Then the adversary can ask the following queries:

1) `GetRandChal()` – It picks a new $x \xleftarrow{\$} \mathcal{C}$, marks it *fresh* and outputs it;
2) `GetResp`(x^*) – It increments ctr if $x^* \neq x$;
   - If $\mathsf{ctr} \leq \eta$ and $x^* \in \mathcal{C}$, it outputs $F_K(x^*)$ and marks x as *unfresh*;
   - Otherwise, it outputs $\perp$;
3) `TestResp`(r) –
   - If $F_K(x) = r$ and x is *fresh*, the adversary *wins*;
   - If $F_K(x) = r$ and x is *unfresh*, it outputs 1;
   - Otherwise, it outputs 0.

Just like the η-unforgeability experiment, the above oracle calls are sequential (similar to Figure 1), starting with a `GetRandChal`-query. But since non-malleability is assumed, only `GetResp`-queries with inputs different from the current random challenges make the counter increase, and it is never decreased.

The advantage of any adversary $\mathcal{A}$ against the above unforgeability, $\mathsf{Adv}^{\eta\text{-uf-basic}}_F(\mathcal{A})$ is the probability of winning in the above experiment. Such a success indeed means that the adversary found the response for a new random challenge, without having asked for a `GetResp`-query. The parameter η restricts the number of "adaptive" `GetResp` queries that $\mathcal{A}$ can make, where adaptive means "different from the current random challenge".

The resources of the adversary are the polynomial running time and the numbers $q'_{\mathsf{c}}, q'_{\mathsf{r}}, q'_{\mathsf{t}}$ of queries to the above `GetRandChal`, `GetResp` and `TestResp` oracles, respectively. For convenience, denote by q''_{t} the number of `TestResp`-queries such that the current random challenge x is *fresh*. By definition, we have $q'_{\mathsf{r}} \leq q'_{\mathsf{c}}$, $q'_{\mathsf{t}} \leq q'_{\mathsf{c}}$ and $q''_{\mathsf{t}} \leq q'_{\mathsf{c}} - q'_{\mathsf{r}}$.

HC FUNCTION FAMILY SECURITY RESULTS. Under Conjecture 1, one can prove the following unforgeability result about the HC function family.

Theorem 3 (From [4]): Given $\varepsilon, \varepsilon' > 0, t \in \mathbb{N}_+$ and $\delta > (\frac{1}{10} + \varepsilon)^t$, for any probabilistic polynomial time (in $n, q'_{\mathsf{c}}, 1/\varepsilon$) adversary $\mathcal{A}$ against the basic 0-unforgeability security of the HC function family F constructed above using $f = f_{k_1,k_2}$ with

$$q''_{\mathsf{t}} = 1, q'_{\mathsf{c}} \leq \frac{1}{t} \cdot n^{\min\{(k_2+1)/2, k_1+1-\varepsilon'\}} - 1,$$

under Conjecture 1, we have $\mathsf{Adv}^{\mathsf{0\text{-}uf\text{-}basic}}_F(\mathcal{A}) < \delta$.

Note that in the basic 0-unforgeability security game, the adversary learns nothing from `GetResp`(x^*) if x^* is not the current random challenge x. So if $\eta = 0$, the adversary $\mathcal{A}$ is only given random challenge-response pairs.

This result is actually not strictly good-enough, even for our confirmed HAKE. Indeed, if the function does not allow for at least one adaptive query, an attacker could make it in the first exchange (using an infected terminal), then break the unforgeability of the function before the confirmation flow and make the protocol succeed, hence avoiding detection. Thus, we extend the RP-CSP conjecture to allow $\log n$ adaptive "small" challenge-response pairs.

Conjecture 2: Consider the function f_{k_1,k_2} described above, for any $\varepsilon, \varepsilon' > 0, t \in \mathbb{N}_+$ and any probabilistic polynomial time (in n) adversary $\mathcal{A}$, there exists an interger $N \in \mathbb{N}$, such that for all $n > N$, $m_r \leq n^{\min\{(k_2+1)/2, k_1+1-\varepsilon'\}}$ and $m_a \leq t \log n$, we have $\mathsf{Adv}^{\mathsf{adapt}}_{f_{k_1,k_2}}(\mathcal{A}, \varepsilon) = \mathsf{negl}(n)$, where $\mathsf{Adv}^{\mathsf{adapt}}_{f_{k_1,k_2}}(\mathcal{A}, \varepsilon)$ is the probability that $\mathcal{A}$ outputs a mapping σ' that is ε-correlated with the secret mapping σ given m_r random "small" challenge-response pairs and the correct responses to m_a "small" challenges adaptively chosen by $\mathcal{A}$.

Proof: For any adversary $\mathcal{A}$ we can construct an adversary $\mathcal{B}$ such that $\mathsf{Adv}^{\mathsf{adapt}}_{f_{k_1,k_2}}(\mathcal{A}, \varepsilon) \leq 10^{t \log n} \times \mathsf{Adv}^{\mathsf{rand}}_{f_{k_1,k_2}}(\mathcal{B}, \varepsilon)$.

$\mathcal{B}$ simulates $\mathcal{A}$'s view by providing $\mathcal{A}$ with the given m_r random "small" challenge-response pairs and randomly guessing the responses to the m_a ($\leq t \log n$) adaptive "small"

challenges. The probability of correctly guessing all adaptive ones is $10^{-t \log n}$ (refer to the construction of f_{k_1,k_2}), hence the above advantage reduction. One should note that $10^{t \log n} \times \mathsf{negl}(n) = \mathsf{negl}(n)$ and $\mathcal{B}$'s running time is polynomial in n. ■

Under this extended conjecture, one can prove the following stronger unforgeability result about the HC function family, which "almost" suits our confirmed HAKE (see Fig. 3):

Theorem 4: Given $\varepsilon, \varepsilon' > 0, t \in \mathbb{N}_+$ and $\delta > (\frac{1}{10} + \varepsilon)^t$, for any probabilistic polynomial time (in $n, q'_{\mathsf{c}}, 1/\varepsilon$) adversary $\mathcal{A}$ against the basic η-unforgeability security of the HC function family F constructed above using $f = f_{k_1,k_2}$ with

$$\eta \leq \log n, q'_{\mathsf{c}} \leq \frac{1}{t} \cdot n^{\min\{(k_2+1)/2, k_1+1-\varepsilon'\}} - 1,$$

under Conjecture 2, we have $\mathsf{Adv}_F^{\eta\text{-uf-basic}}(\mathcal{A}) < q''_{\mathsf{t}} \cdot \delta$.

Proof: The proof is almost the same as that of Theorem 3. Informally, we need to show that any adversary $\mathcal{A}$ that breaks the basic η-unforgeability security of the HC function family can also "recover" the secret mapping σ in Conjecture 2. The reader can refer to the proof of Theorem 5 in [4], which we call the "HCP" proof below, for the details.

Nevertheless, here the theorem differs from Theorem 3 in several aspects. First, $\mathcal{A}$ can adaptively select $t \log n$ "small" challenges to get the correct responses, while in Theorem 3 only random ones are allowed. But having adaptive queries does not affect the HCP proof because it only uses $\mathcal{A}$ as a blackbox to predict the responses to any t "small" challenges. Second, we apply an union bound of q''_{t} queries to the final advantage. ■

Remark 3: In the above theorem $\varepsilon, \varepsilon'$ are almost 0. We can set $n = 100$, $k_1 = 1$, $k_2 = 3$ and $t = 5$, then $\eta \leq 6$, $q'_{\mathsf{c}} \leq n^2/t - 1 \approx 2000$ and $\mathsf{Adv}_F^{\eta\text{-uf-basic}}(\mathcal{A}) < q''_{\mathsf{t}} \cdot 10^{-t} \leq 1/50$.

We believe a similar theorem holds for $\mathsf{Adv}_F^{\eta\text{-uf}}$ (by replacing the oracles with those in the 2-party η-unforgeability experiment), which our HAKE security can rely on. The intuition is as follows. With the HC function family instantiation described in this section, a $\texttt{GetResp}(x^*)$ query in the η-unforgeability experiment should not have x^* too "far" from the random challenge x output by the latest `GetRandChal` query. Otherwise, it is very unlikely for the adversary to guess correctly in the `TestResp` query. But the adversary can modify x a little bit to guess the correct response with a smaller failure probability. This is the difference between the two unforgeability notions: the basic one does not tolerate any malleability, whereas the other can exploit malleability. Because of the size of the challenge space, that has to be quite large (it is essentially $n^{t(10+k_1+k_2)}$, and thus 2^{465}, with the above parameters), the number of challenges that are "close" to any random challenge accounts for a tiny proportion. Thus, the adversary should not get much help from such "nearly random" challenges. Besides, such queries risk increasing the counter in the `GetResp` oracle without extracting much useful information. In addition, the two memory slots will not increase much the advantage of an adversary, and so $\mathsf{Adv}_F^{\eta\text{-uf-basic}}$ and $\mathsf{Adv}_F^{\eta\text{-uf}}$ should be quite close for this specific HC function family instantiation. We leave further studies of security of the HC function family from [4] to future works.

VI. Device-Assisted HAKE Protocols

In this section, we take a step back from Only-Human HC function family to allow the use of an additional device that will perform the computations in place of the human. In this setting, the HC function family can be quite powerful and thus resist to many adaptive queries. We consider it in two scenarios: first in a similar context as the Basic Generic HAKE, where one can enter a challenge onto the device to get the response; and second, a time-based token, that outputs the response every timeframe, with the time as the challenge (without having the user to enter it).

A. Simplified Basic HAKE

According to the security proof of the Basic Generic HAKE, the PAKE has to be instantiated with a UC-Secure protocol, which turns out to be quite costly. Indeed, the only efficient scheme that achieves this security level is the encrypted key exchange protocol (EKE) [1]. However, the proof holds in the Ideal-Cipher model, for a symmetric blockcipher that should only output elements in the Diffie-Hellman group. In practice, the best way to do it is to iterate a large blockcipher until one falls in the group. First, a large blockcipher from a hash function (modeled as a random oracle) has fueled a whole line of works [43]–[45], and is nevertheless already quite costly: at the time of writing, at least 8-round Feistel network is required [45], with an impossibility result below 6 [43]. Thereafter, additional iterations are required to build a permutation onto the group. This thus eventually corresponds to dozens of hash function evaluations.

Looking back at the construction, using a full PAKE seems anyway as a bit of an overkill since the ephemeral secrets are only used once, and need not to be kept secret afterwards. We present in the full version [36] a protocol that uses commitments instead of a full PAKE to achieve better efficiency.

B. Time-Based HAKE

Scenario. In this section, we focus on the particular (but quite usual) case where the physical device does not have a dedicated input but uses time instead to compute its output. More precisely, our protocol considers a device, such as the RSA-SecurId [28] token, that, based on an internal seed (the long-term key K_ℓ), generates a one-time password (the value $F_{K_\ell}(t)$, based on the time period t), and displays it on an LCD-Screen. The password is tied to an internal clock, and changes every τ (e.g. 30s). Note that such a password is already *human readable* and *human writable*, hence it satisfies our human-compatible communications.

Building on the security model presented in Section II-B, we now consider time as a variable, that is to be segmented into timeframes (each spanning τ seconds). We then number those timeframes and associate to each message sent between T and S this number, representing the fact that each party can

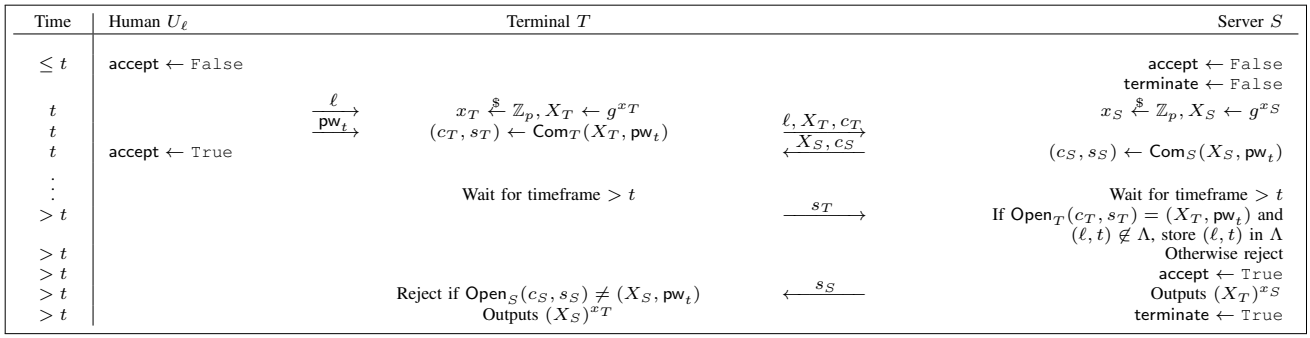

Fig. 4. Time-Based Device-Assisted HAKE Construction

measure time and identify the timeframe in which the message was sent.

Since the one-time passwords are generated by a secure device implementing F_{K_ℓ}, we can make the assumption that, for each timeframe, the output is indistinguishable from an element sampled from the distribution $\mathcal{D}$ with entropy greater than D (which increases the advantage of an adversary $\mathcal{A}$ by at most $\mathsf{Adv}_F^{\text{dist-}T}(\mathcal{A})$ after T timeframes).

We rely on the requirement that any user U_ℓ can only make use of one terminal during a timeframe. That is, he may not attempt to authenticate using more than one terminal in a single time period.

PROTOCOL. We now propose a protocol for time-based device-assisted HAKE. It is presented on Fig. 4. As in the previous Simplified Basic HAKE, it makes use of a commitment scheme on top of the unauthenticated Diffie-Hellman scheme to perform authentication.

The commitment scheme $\mathcal{CS}$ is initialized twice, with two independent setup, leading to $\mathsf{Com}_T/\mathsf{Open}_T$ and $\mathsf{Com}_S/\mathsf{Open}_S$, each of them being used for the commitments generated by the terminal and the server, respectively. We also setup a group $\mathbb{G}$ of prime order p in which the discrete logarithm problem is believed to be hard. Let g be a generator of $\mathbb{G}$.

The protocol itself is split into two parts: the *commitment* phase which must happen during a timeframe t (that we will call the *session timeframe*) and the *verification* phase, that must happen later than the session timeframe.

This delay is a clear limitation on the total speed of the protocol, which on average will take $\tau/2$. It will however prove necessary, as it allows $F_{K_\ell}(t)$ to be revealed without compromising the security of the scheme, therefore building on the one-time specificity of the password. To enforce a unique session in a timeframe, the server will not accept to run several sessions within the same timeframe, with the same user, as the latter should not do it anyway (see above). This would thus come from an adversary, and then allowing multiple sessions in a timeframe t can compromise other sessions in the same timeframe when $\mathcal{F}_{K_\ell}(t)$ is revealed.

It is interesting to note that this protocol uses the time period t as the HAKE challenge (the challenge is a counter) and the one-time password ($F_{K_\ell}(t)$) read from the device as the human's response. Therefore, partnering between U and S is entirely determined at the end of the session timeframe t.

SECURITY ANALYSIS. In the security analysis, as in Section IV, we only consider static *compromises*. Hence $\texttt{Compromise}(j)$ can only be the first oracle query of a session, and $\texttt{Infect}(j)$ can only affect *compromised* sessions. Since *compromises* are known before the first flow and partnering between Human and Server is determined at the end of timeframe t, this means that *freshness* itself can be perfectly ascertained in any timeframe $> t$.

The security of our protocol heavily relies on the strong-security of the commitment scheme (see Section III and the full version [36]).

Theorem 5: Consider the Time-Based Device-Assisted HAKE protocol defined in Fig. 4. Let $\mathcal{A}, \mathcal{A}'$ be an adversaries against the privacy and user authenticity security games with static compromises, running within time t and using less than n_{serv} non-passive sessions against the server oracle, n_{term} non-passive sessions against the terminal oracle, $n_{\mathsf{total}} > n_{\mathsf{term}} + n_{\mathsf{serv}}$ total sessions and $T < n_{\mathsf{total}}$ unique timeframes. Then there exist an adversary $\mathcal{B}_1$ against the indistinguishability of the password-distribution $\mathcal{D}$ running in time t, an adversary $\mathcal{B}_3$ against the DDH experiment running in time $t+8n_{\mathsf{total}}\tau_{\mathsf{exp}}$, and adversary $\mathcal{B}_2$ against the commitment scheme running in time t:

$$\begin{aligned}\mathsf{Adv}^{\mathsf{priv}}_{\mathsf{HAKE}}(\mathcal{A}) \leq (n_{\mathsf{serv}} + n_{\mathsf{term}}) \cdot 2^{-D} + \mathsf{Adv}^{\text{dist-}T}_F(\mathcal{B}_1) \\ \mathsf{Adv}^{\mathsf{ind}}_{\mathsf{DDH}}(\mathcal{B}_3) + (n_{\mathsf{total}} + 3) \cdot \mathsf{Adv}_{\mathcal{CS}}(\mathcal{B}_2), \\ \mathsf{Adv}^{\text{u-auth}}_{\mathsf{HAKE}}(\mathcal{A}') \leq (n_{\mathsf{serv}} + n_{\mathsf{term}}) \cdot 2^{-D} + \mathsf{Adv}^{\text{dist-}T}_F(\mathcal{B}_1) \\ (n_{\mathsf{total}} + 2) \cdot \mathsf{Adv}_{\mathcal{CS}}(\mathcal{B}_2),\end{aligned}$$

with τ_{exp} the time necessary to exponentiate one group element, and n_{total} the global number of sessions.

The proof is in the full version [36].

Remark 4: Note that the Time-Based Device-Assisted HAKE only achieves user authentication in our setting, since *server authentication* requires the server identity to be approved by the human in our setting (the terminal could be *infected* so it cannot be relied on). A similar approach to the one of the Confirmed HAKE could be used to achieve a full mutual authentication.

Scheme	Flows	Terminal		Server		Communication
		expon.	$\mathcal{H}$ eval.	expon.	$\mathcal{H}$ eval.	complexity
`1(SPAKE1)` [35], [46]	4	3	1	3	1	4λ
This work	4	2	2	2	2	10λ

TABLE I
PERFORMANCE OF THE TIME-BASED DEVICE-ASSISTED HAKE

PERFOMANCES. We offer in Table I a comparison (in terms of numbers of flows, exponentiations, $\mathcal{H}$ evaluations and overall communication complexity) of the performances of our HAKE protocol with the one-time PAKE `1(P)` construction of [35], instantiated with `SPAKE1` from [46] as a reference.
Since `SPAKE1` is also proven in the random oracle model, it is fair to use the efficient commitment scheme described in Section III. We do not include the redundant X_P in s_P (it is transmitted at the commitment stage) for the communication complexity, and for a security parameter λ, we assume the group elements to be encoded into 2λ-long bit-strings.
While our communication complexity is higher, the computational load is reduced by 30% from [35] with the most efficient PAKE. Relaxing the PAKE security properties allows a significant gain from the complexity point of view.

VII. CONCLUSION

We proposed the first user authenticated key exchange protocols which can tolerate corrupted terminals: if a user happens to log in to a server from a terminal that has been fully compromised, then the other past and future user's sessions initiated from honest terminals stay secure. We formalized security for Human Authenticated Key Exchange (HAKE) protocols and proposed generic constructions based on human-compatible (HC) function families or small auxiliary devices such as RSA SecurID, password-authenticated key exchange (PAKE), commitment, and authenticated encryption. We analyzed security of our HAKE protocols and discussed their instantiations. We left several interesting open problems and believe that our work will promote further developments in the area of human-oriented cryptography.

ACKNOWLEDGMENTS

We thank the reviewers for insightful comments. Alexandra Boldyreva and Shan Chen are supported in part by the National Science Foundation under Grants No. CNS-1318511 and CNS-1422794. Pierre-Alain Dupont and David Pointcheval are supported in part by the European Research Council under the European Community's Seventh Framework Programme (FP7/2007-2013 Grant Agreement no. 339563 – CryptoCloud).

REFERENCES

[1] S. M. Bellovin and M. Merritt, "Encrypted key exchange: Password-based protocols secure against dictionary attacks," in *1992 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 1992, pp. 72–84.

[2] M. Bellare, D. Pointcheval, and P. Rogaway, "Authenticated key exchange secure against dictionary attacks," in *EUROCRYPT 2000*, ser. LNCS, B. Preneel, Ed., vol. 1807. Springer, Heidelberg, May 2000, pp. 139–155.

[3] R. Canetti, "Universally composable security: A new paradigm for cryptographic protocols," in *42nd FOCS*. IEEE Computer Society Press, Oct. 2001, pp. 136–145.

[4] J. Blocki, M. Blum, A. Datta, and S. Vempala, "Towards human computable passwords," *arXiv preprint arXiv:1404.0024v4*, 2016.

[5] V. Feldman, W. Perkins, and S. Vempala, "On the complexity of random satisfiability problems with planted solutions," in *47th ACM STOC*, R. A. Servedio and R. Rubinfeld, Eds. ACM Press, Jun. 2015, pp. 77–86.

[6] M. Bellare, C. Namprempre, D. Pointcheval, and M. Semanko, "The one-more-RSA-inversion problems and the security of Chaum's blind signature scheme," *Journal of Cryptology*, vol. 16, no. 3, pp. 185–215, Jun. 2003.

[7] L. von Ahn, M. Blum, N. J. Hopper, and J. Langford, "CAPTCHA: Using hard AI problems for security," in *EUROCRYPT 2003*, ser. LNCS, E. Biham, Ed., vol. 2656. Springer, Heidelberg, May 2003, pp. 294–311.

[8] T. Katoh and H. Imai, "An application of visual secret sharing scheme concealing plural secret images to human identification scheme," in *Proc. of SITA*, vol. 96, 1996, pp. 661–664.

[9] M.-R. Kim, J.-H. Park, and Y. Zheng, "Human-machine identification using visual cryptography," in *Proceedings of the 6th IEEE International Workshop on Intelligent Signal Processing and Communication Systems*, 1998, pp. 178–182.

[10] I. Jermyn, A. Mayer, F. Monrose, M. K. Reiter, and A. D. Rubin, "The design and analysis of graphical passwords," in *Proceedings of the 8th Conference on USENIX Security Symposium - Volume 8*, ser. SSYM'99. Berkeley, CA, USA: USENIX Association, 1999, pp. 1–1. [Online]. Available: http://dl.acm.org/citation.cfm?id=1251421.1251422

[11] R. Dhamija and A. Perrig, "Déjà vu: A user study using images for authentication," in *Proceedings of the 9th Conference on USENIX Security Symposium - Volume 9*, ser. SSYM'00. Berkeley, CA, USA: USENIX Association, 2000, pp. 4–4. [Online]. Available: http://dl.acm.org/citation.cfm?id=1251306.1251310

[12] J. Thorpe and P. C. van Oorschot, "Graphical dictionaries and the memorable space of graphical passwords," in *Proceedings of the 13th USENIX Security Symposium, August 9-13, 2004, San Diego, CA, USA*, M. Blaze, Ed. USENIX, 2004, pp. 135–150. [Online]. Available: http://www.usenix.org/publications/library/proceedings/sec04/tech/thorpe.html

[13] R. Biddle, S. Chiasson, and P. Van Oorschot, "Graphical passwords: Learning from the first twelve years," *ACM Comput. Surv.*, vol. 44, no. 4, pp. 19:1–19:41, Sep. 2012. [Online]. Available: http://doi.acm.org/10.1145/2333112.2333114

[14] D. Weinshall and S. Kirkpatrick, "Passwords you'll never forget, but can't recall," in *CHI '04 Extended Abstracts on Human Factors in Computing Systems*, ser. CHI EA '04. New York, NY, USA: ACM, 2004, pp. 1399–1402. [Online]. Available: http://doi.acm.org/10.1145/985921.986074

[15] T. Matsumoto and H. Imai, "Human identification through insecure channel," in *EUROCRYPT'91*, ser. LNCS, D. W. Davies, Ed., vol. 547. Springer, Heidelberg, Apr. 1991, pp. 409–421.

[16] C.-H. Wang, T. Hwang, and J.-J. Tsai, "On the Matsumoto and Imai's human identification scheme," in *EUROCRYPT'95*, ser. LNCS, L. C. Guillou and J.-J. Quisquater, Eds., vol. 921. Springer, Heidelberg, May 1995, pp. 382–392.

[17] T. Matsumoto, "Human-computer cryptography: An attempt," in *ACM CCS 96*. ACM Press, Mar. 1996, pp. 68–75.

[18] X.-Y. Li and S.-H. Teng, "Practical human-machine identification over insecure channels," *Journal of Combinatorial Optimization*, vol. 3, no. 4, pp. 347–361, 1999.

[19] S. Dziembowski, "How to pair with a human," in *SCN 10*, ser. LNCS, J. A. Garay and R. D. Prisco, Eds., vol. 6280. Springer, Heidelberg, Sep. 2010, pp. 200–218.

[20] A. Juels and S. A. Weis, "Authenticating pervasive devices with human protocols," in *CRYPTO 2005*, ser. LNCS, V. Shoup, Ed., vol. 3621. Springer, Heidelberg, Aug. 2005, pp. 293–308.

[21] J. Bringer and H. Chabanne, "Trusted-HB: a low-cost version of HB+ secure against man-in-the-middle attacks," Cryptology ePrint Archive, Report 2008/042, 2008, http://eprint.iacr.org/2008/042.

[22] J. Bringer, H. Chabanne, and E. Dottax, "HB++: a lightweight authentication protocol secure against some attacks," in *Security, Privacy and Trust in Pervasive and Ubiquitous Computing, 2006. SecPerU 2006. Second International Workshop on*. IEEE, 2006, pp. 28–33.

[23] J. Munilla and A. Peinado, "HB-MP: A further step in the HB-family of lightweight authentication protocols," *Computer Networks*, vol. 51, no. 9, pp. 2262–2267, 2007.

[24] X. Leng, K. Mayes, and K. Markantonakis, "HB-MP+ protocol: An improvement on the HB-MP protocol," in *RFID, 2008 IEEE International Conference on*. IEEE, 2008, pp. 118–124.

[25] K. A. Khoureich, "hHB: a harder HB+ protocol," Cryptology ePrint Archive, Report 2014/562, 2014, http://eprint.iacr.org/2014/562.

[26] ——, "Light-hHB: A new version of hHB with improved session key exchange," Cryptology ePrint Archive, Report 2015/713, 2015, http://eprint.iacr.org/2015/713.

[27] N. J. Hopper and M. Blum, "Secure human identification protocols," in *ASIACRYPT 2001*, ser. LNCS, C. Boyd, Ed., vol. 2248. Springer, Heidelberg, Dec. 2001, pp. 52–66.

[28] "RSA SecurId Hardware Tokens," RSA Security, https://www.rsa.com/en-us/products-services/identity-access-management/securid/hardware-tokens.

[29] N. Haller, "The s/key one-time password system," Internet Requests for Comments, IETF, RFC 1760, February 1995, http://tools.ietf.org/html/rfc1760.

[30] N. Haller, C. Metz, P. Nesser, and M. Straw, "A one-time password system," Internet Requests for Comments, IETF, RFC 2289, February 1998, http://tools.ietf.org/html/rfc2289.

[31] M. Nystroem, "The EAP protected one-time password protocol (EAP-POTP)," Internet Requests for Comments, IETF, RFC 4793, February 2007, http://tools.ietf.org/html/rfc4793.

[32] D. M'Raihi, M. Bellare, F. Hoornaert, D. Naccache, and O. Ranen, "HOTP: An HMAC-based one-time password algorithm," Internet Requests for Comments, IETF, RFC 4226, December 2005, https://tools.ietf.org/html/rfc4226.

[33] D. M'Raihi, S. Machani, M. Pei, and J. Rydell, "TOTP: Time-based one-time password algorithm," Internet Requests for Comments, IETF, RFC 6238, May 2011, https://tools.ietf.org/html/rfc6238.

[34] "Google Authenticator," Google, Inc., https://support.google.com/accounts/answer/1066447?hl=en&rd=1.

[35] K. G. Paterson and D. Stebila, "One-time-password-authenticated key exchange," in *ACISP 10*, ser. LNCS, R. Steinfeld and P. Hawkes, Eds., vol. 6168. Springer, Heidelberg, Jul. 2010, pp. 264–281.

[36] A. Boldyreva, S. Chen, P.-A. Dupont, and D. Pointcheval, "Human computing for handling strong corruptions in authenticated key exchange," Cryptology ePrint Archive, Report 2017/559, 2017, http://eprint.iacr.org/2017/559.

[37] M. Bellare and P. Rogaway, "Random oracles are practical: A paradigm for designing efficient protocols," in *ACM CCS 93*, V. Ashby, Ed. ACM Press, Nov. 1993, pp. 62–73.

[38] R. Canetti, S. Halevi, J. Katz, Y. Lindell, and P. D. MacKenzie, "Universally composable password-based key exchange," in *EUROCRYPT 2005*, ser. LNCS, R. Cramer, Ed., vol. 3494. Springer, Heidelberg, May 2005, pp. 404–421.

[39] M. Abdalla, D. Catalano, C. Chevalier, and D. Pointcheval, "Efficient two-party password-based key exchange protocols in the UC framework," in *CT-RSA 2008*, ser. LNCS, T. Malkin, Ed., vol. 4964. Springer, Heidelberg, Apr. 2008, pp. 335–351.

[40] M. Bellare and C. Namprempre, "Authenticated encryption: Relations among notions and analysis of the generic composition paradigm," in *ASIACRYPT 2000*, ser. LNCS, T. Okamoto, Ed., vol. 1976. Springer, Heidelberg, Dec. 2000, pp. 531–545.

[41] P. Rogaway, M. Bellare, J. Black, and T. Krovetz, "OCB: A block-cipher mode of operation for efficient authenticated encryption," in *ACM CCS 01*. ACM Press, Nov. 2001, pp. 196–205.

[42] V. Feldman, W. Perkins, and S. Vempala, "On the complexity of random satisfiability problems with planted solutions," in *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing*. ACM, 2015, pp. 77–86.

[43] J.-S. Coron, J. Patarin, and Y. Seurin, "The random oracle model and the ideal cipher model are equivalent," in *CRYPTO 2008*, ser. LNCS, D. Wagner, Ed., vol. 5157. Springer, Heidelberg, Aug. 2008, pp. 1–20.

[44] T. Holenstein, R. Künzler, and S. Tessaro, "The equivalence of the random oracle model and the ideal cipher model, revisited," in *43rd ACM STOC*, L. Fortnow and S. P. Vadhan, Eds. ACM Press, Jun. 2011, pp. 89–98.

[45] Y. Dai and J. P. Steinberger, "Indifferentiability of 8-round feistel networks," in *CRYPTO 2016, Part I*, ser. LNCS, M. Robshaw and J. Katz, Eds., vol. 9814. Springer, Heidelberg, Aug. 2016, pp. 95–120.

[46] M. Abdalla and D. Pointcheval, "Simple password-based encrypted key exchange protocols," in *CT-RSA 2005*, ser. LNCS, A. Menezes, Ed., vol. 3376. Springer, Heidelberg, Feb. 2005, pp. 191–208.