LATKE: A Framework for Constructing Identity-Binding PAKEs

Jonathan Katz
University of Maryland
jkatz2@gmail.com

Michael Rosenberg
University of Maryland
micro@cs.umd.edu

Version 0.5, March 16, 2024

Abstract

Motivated by applications to the internet of things (IoT), Cremers, Naor, Paz, and Ronen (Crypto '22) recently considered a setting in which multiple parties share a common password and want to be able to securely authenticate to each other. They observed that using standard password-authenticated key exchange (PAKE) protocols in this setting allows for *catastrophic impersonation* attacks whereby compromise of a single party allows an attacker to impersonate any party to any other. To address this, they proposed the notion of *identity-binding PAKE* (iPAKE) and showed constructions of iPAKE protocols CHIP and CRISP.

In this work we present LATKE, a new framework for iPAKE that allows us to construct protocols offering features beyond what CHIP and CRISP achieve. In particular, we can instantiate the components of our framework to yield an iPAKE protocol with post-quantum security and *identity concealment*, where one party hides its identity until it has authenticated the other. To our knowledge, this is the first iPAKE protocol with either property. We show that the iPAKEs produced by LATKE UC-realize a slightly weakened version of the original iPAKE functionality in the adaptive corruption model with erasure and programmable random oracles.

To demonstrate the concrete efficiency of our framework, we implement various instantiations and compare the resulting protocols to CHIP when run on commodity hardware. We find some pre-quantum instantiations have computation cost within 5% of CHIP and with a communication overhead of 324B, and one post-quantum instantiation achieves computation cost within 3% of CHIP with a communication overhead of 3kB.

Keywords: key agreement, password-based cryptography, IoT, post-quantum cryptography

1 Introduction

A human-entered password is one of the most common methods of authentication today. Passwords are used for logging into websites, performing secure file transfers, connecting to WiFi, connecting to mesh networks, storing encrypted backups, and logging into remote servers.

Special protocols are necessary to handle password-based authentication. Many of the applications above require an encrypted channel to avoid sending an unencrypted password over an untrusted network. For some protocols, such as HTTPS, this is straightforward, since the server has a known public key which can be used to bootstrap a secure session. But other use cases, such as secure point-to-point file transfer like Magic Wormhole [mag], have no such infrastructure upon which to build. To communicate securely, two clients need some way of using their mutual knowledge of a low-entropy password to establish a high-entropy shared secret, over an insecure channel. This is precisely the setting that a *password authenticated key exchange* (PAKE) algorithm solves.

Since its introduction by Bellovin and Meritt [BM92], PAKE has been extended to offer stronger security in more diverse scenarios. Some widely deployed internet of things (IoT) protocols already use PAKE and its extensions for authentication between devices [Thr15, All22a]. Similarly, the WiFi specification now includes the SAE PAKE [Har08] for mutual authentication of stations in a network [All22b, wif21].

For IoT and mesh networks in general, the strongest notion of PAKE that exists is *identity-binding PAKE* (iPAKE), introduced by Cremers, Naor, Paz, and Ronen [CNPR22]. This form of PAKE allows each party to bind to its key material a short identity string. If an adversary compromises an iPAKE party, they can impersonate that party, but they cannot do much more. In particular, if the adversary wants to determine the password, or impersonate another party with the same password, they will have to mount a brute-force attack on the password.

We identify shortcomings in existing iPAKE constructions and provide a generic, efficient iPAKE framework that addresses these shortcomings at the cost of a concretely small overhead in runtime and communication size. We will prove LATKE iPAKEs secure in a security model slightly weaker than that of CHIP, and will argue that this model is a natural one to use for any iPAKE.

1.1 Background

PAKE. The problem of communicating securely using mutual knowledge of a low-entropy password was first addressed by Bellovin and Merritt [BM92], in the form of PAKE. PAKE offers the security guarantees that 1) a passive adversary learns nothing about the password; 2) an active adversary cannot guess the password more than once per session; and 3) barring a correct password guess in an active attack, the resulting session key has high entropy and is known only to the participants. Unlike most other cryptographic constructions, the probability of an adversary's success is not bounded by a negligible function, but rather by its ability to guess the password after some number of attempts. This is as good a guarantee as is possible, given the constraint that honest users must be able to establish a key using only the password.

PAKE is not ideal for all use cases. It requires both parties to have a plaintext copy of the password on hand. This works in short-lived interactive protocols with humans in the loop such as file transfer, or with a smartcard and a reader in the loop, such as e-passport NFC protocols (where the machine-readable zone is the password) [Hv22]. But this breaks down when one party is a server that runs for long periods without interaction, e.g., a password-protected backup server. While the client can type a password at authentication time, the long-lived server needs to store something. If it stores the password, then an adversary who compromises the server learns the password.

aPAKE $\supset$ PAKE. *Augmented password-authenticated key exchange* (aPAKE)—first proposed by Jablon [Jab97] and later formalized by Gentry, MacKenzie, and Ramzan [GMR06]—extends PAKE to afford more security to the server in the above scenarios. aPAKE protocols work by having the server store a per-client *password file*, which is computed as some hard-to-invert function of the password. If an adversary compromises the server and retrieves the password file, they can impersonate the server to a client. To impersonate the client, the adversary can do no better than a brute-force attack, specifically, guessing at the password until its corresponding password file matches the retrieved one. Many commonly used protocols rely on aPAKE, including iCloud key recovery [App, p. 132], WhatsApp message history recovery [DFG+23], and 1Password user authentication [Fil18].

A related security notion is *strong aPAKE* (saPAKE). With an ordinary aPAKE, an adversary can precompute large tables of password hashes so that, when they compromise a server, their brute-force attack becomes a simple table lookup. More complex data structures, called *rainbow tables*, offer tunable time-memory tradeoffs for such attacks [Oec03]. Strong aPAKE—first described by Jarecki, Krawczyk, and Xu [JKX18]—prevents precomputation attacks by using a secret (even to active adversaries) blinding factor to compute the password file. In their paper, they describe generic saPAKE compiler which requires only an oblivious pseudorandom function (OPRF) and an aPAKE.

Again, though, (s)aPAKE is not appropriate for all use cases. While the server can run the protocol using just the password file, the client must have the plaintext password. Thus, we have the same problem as before: if the client stores its password, and the client is compromised, then its password is leaked. An adversary can then use a leaked password to impersonate any client or server with the same password. Concretely, this is a problem for devices connecting to WiFi[1] or for re-authenticating to (s)aPAKE-based services without prompting a human.

dPAKE $\supset$ aPAKE. *Doubly augmented password-authenticated key exchange* (dPAKE)—first described by Hamburg [Ham15]—is an aPAKE where the client also stores a password file, rather than a cleartext password. Thus, if *either*

[1]One possible solution is to have every pair of client and access point establish a long-term key in their first session, which they will use to re-authenticate themselves in the future. But this comes with the requirement that a mesh of WiFi access points using Wireless Distribution System (WDS), i.e., pretending to be a single access point, will constantly sync the pairwise keys with each other.

party is compromised, the adversary cannot do better than either impersonating that party or else performing a brute-force search for the password. dPAKE also has a strong variant, sdPAKE that, like saPAKE, prevents precomputation attacks. As with aPAKE, it is possible to generically construct sdPAKE from dPAKE using an OPRF.

At the time of writing, (s)dPAKE remains a folklore definition with no formal security definition. But it is a useful conceptual extension of aPAKE, and has been proposed as a replacement for SAE for WiFi stations [Tho22b].

Finally, even dPAKE is insufficient for some use cases. dPAKE has a coarse notion of identity: every party is either a client, indistinguishable from any other client, or a server, indistinguishable from any other server. This works in the case of WiFi, where clients do not need to be distinguished from one another, nor do access points. But this is unacceptable in mesh networks, where each device may have unique capabilities and permissions.

iPAKE $\supset$ dPAKE. *Identity-binding password-authenticated key exchange* (iPAKE)—first described by Cremers, Naor, Paz, and Ronen [CNPR22]—is a strengthening of dPAKE in which each party's password file is bound to an identity string of their choosing, and can be used to authenticate with any other party. Similar to dPAKE, an attacker who compromises a party can impersonate that party or be force to mount a brute-force attack on the password to create their own password file.

The authors of [CNPR22] define iPAKE in the Universal Composability (UC) framework [Can01] and construct CHIP, an efficient iPAKE protocol. The CHIP construction is generic, requiring only a PAKE and an identity-based key exchange protocol (IBKE) with certain properties (explained in the next paragraph). Finally, the paper defines a notion of *strong iPAKE (siPAKE)* with the same precomputation prevention properties as the aforementioned strong constructions. The authors provide an efficient siPAKE protocol, called CRISP, and prove it secure in the UC framework using the generic group model.

1.2 Limitations of Existing iPAKEs and dPAKEs

CHIP's IBKE requirements limit the variety of scenarios in which the iPAKE can be deployed. CHIP needs the underlying IBKE to be have key compromise impersonation resistance (KCIR) and be msk-*independent*, i.e., the message flow of the protocol must be statistically independent of msk. While there are a handful of KCIR, msk-independent IBKE protocols [Oka88, Gün90, FG10, Shi03, Wan13, CC07b, CC07a], they are insufficient to give CHIP some desirable properties.

Quantum resistance. All the aforementioned IBKEs rely on ordinary or bilinear Diffie-Hellman-type assumptions. Hence, there is no clear way to build iPAKE from *post-quantum assumptions*, i.e., cryptographic assumptions that plausibly hold for quantum computers. As the horizon for cryptographically relevant quantum computers shortens [You22], it is important to have readily available alternatives relying on different assumptions. This is especially relevant in systems that handle data intended to be confidential for a long period of time, and in systems that have a long migration or updating period for their devices and firmware, as is common in industrial settings [Pau22].

Beyond post-quantum iPAKE, there also no clear way to construct an *efficient* post-quantum dPAKE. The OPAQUE aPAKE compiler (not to be confused with the same paper's saPAKE compiler) [JKX18]—doubly augmented to be a dPAKE [Ham15]—can be instantiated with entirely post-quantum primitives, namely a post-quantum authenticated key exchange (AKE) protocol, and a post-quantum oblivious pseudorandom function (OPRF). However, the only options for post-quantum OPRF at the time of writing are either broken [BKW20, BKM$^+$21], or require two to four orders of magnitude greater computation time and communication overhead than Diffie-Hellman-based OPRFs [FOO23, Dod23], or have unknown concrete runtime [Bas22, Bas23]. Thus, practical use is limited.

Identity concealment. Another property not present in the listed IBKEs is that of *identity concealment*, i.e., one of the parties remains anonymous until the other party has successfully authenticated itself, and both parties remain anonymous to a passive eavesdropper. This idea was introduced in the SIGMA-I/SIGMA-R protocols [Kra03] and is available in IKEv2 which is used in the widely deployed IPSec VPN protocol. If an iPAKE device responds to every incoming request with its identity string, an attacker can *wardrive*, i.e., move through a geographical area, attempt to connect to all the devices within range, and map the network's topology [SPT13]. This may be problematic in the case of industrial IoT, e.g., leaking which building a specific sensor or controller resides in.

1.3 Our contributions

We present $\underline{\text{L}}$ow-b$\underline{\text{a}}$rrier identity-$\underline{\text{t}}$ied password-authenticated $\underline{\text{k}}$ey $\underline{\text{e}}$xchange (LATKE), a highly flexible framework for building identity-binding PAKE (iPAKE). LATKE is built from an identity-based authenticated key exchange (IBKE) protocol with key-compromise impersonation resistance (KCIR) and full forward secrecy (full FS), and an ordinary PAKE. These IBKE assumptions are achieved by a wide range of IBKEs, as opposed to the msk-independence assumption that CHIP requires of its IBKE. In fact, as we show, it is possible to construct a LATKE-compatible IBKE using only a signature scheme and an authenticated key exchange (AKE) protocol.

Further extending the compatibility of LATKE with existing IBKEs, we split the framework into two variants, LATKE$^{\text{pre}}$ and LATKE$^{\text{post}}$. The former is intended to be used for *pre-specified peer* IBKEs, i.e., ones wherein the peers know the identity of their intended partner in advance. The latter is intended to be used for *post-specified peer* IBKEs, i.e., ones wherein that is not the case. Since all identity-concealing key exchange protocols have post-specified peers by definition, this allows us to achieve identity concealment.

Finally, a definitional contribution. In order to build LATKE, we slightly weaken the ideal functionality $\mathcal{F}_{\mathsf{iPAKE}}$ given in [CNPR22]. We define our new functionality $\mathcal{F}^{\text{ocw}}_{\mathsf{iPAKE}}$ and argue for its naturalness.

We prove the iPAKEs our frameworks produce UC-realize $\mathcal{F}^{\text{ocw}}_{\mathsf{iPAKE}}$ under adaptive corruptions in the $\mathcal{F}_{\mathsf{RO}}$-hybrid model with erasure. Since there exist post-quantum AKEs/IBKEs and post-quantum PAKEs, we conclude that LATKE can be instantiated as a post-quantum iPAKE.[2] To the authors' knowledge, this is the first description or implementation of a post-quantum or identity-concealing iPAKE. In addition, our construction also represents the most efficient post-quantum dPAKE, since the only known post-quantum dPAKE currently requires post-quantum OPRF, whose shortcomings were described above.

Finally, to demonstrate concrete efficiency, we instantiate LATKE and CHIP using various building blocks—pre-quantum, post-quantum, and identity-concealing—and benchmark them on a commodity WiFi router at the 128-bit security level. We find some pre-quantum instantiations have computation cost within 5% of CHIP and with a communication overhead of 324B, and one post-quantum instantiation achieves computation cost within 3% of CHIP with a communication overhead of 3kB.

2 Preliminaries

In this section we present the notation, definitions, and security models necessary for the construction of LATKE.

2.1 Notation

We write probabilistic algorithms as $\mathsf{Alg}(x; r)$ where x denotes the input and r denotes the random coins. We write $y \leftarrow \mathsf{Alg}(x)$ to denote sampling a value from a probabilistic algorithm, and $x \leftarrow\!\!\$\ S$ to denote sampling a value uniformly from a set S. For our security proofs we will consider probabilistic polynomial time (PPT) adversaries, and denote them with calligraphic letters $\mathcal{A}$. We denote the output x, x' of either side of an interactive protocol between parties A, B by $(x, x') \leftarrow (A \Leftrightarrow B)$. For interactive protocols, we say a *protocol round* is the set of all messages that can be sent in parallel from any point in the protocol [Gon93].[3] We use λ to denote the security parameter.

In our pseudocode for ideal functionalities, **retrieve** denotes retrieval of a record with a specific marking, which, upon failure, returns "no record". **assert** means the functionality ignores the query if the specified condition is not met. We use framed boxes to mark where we elide full subprotocol executions. We will write $\boxed{\mathsf{PAKE}_{\mathsf{sid}\|\mathsf{ssid}}}$ to denote the execution of a PAKE protocol, with session identifier $\mathsf{sid}\|\mathsf{ssid}$.

[2]While LATKE is built from assumptions that are plausibly quantum-hard, its reduction is only stated for classical adversaries. Boneh et al. [BDF+11] show that if a security reduction in the ROM is *history-free*, i.e., oracle queries do not depend on the values of other oracle queries or the query number, then the reduction holds in the quantum ROM as well. We believe our reductions are history-free, but leave the exploration of this path to future work.

[3]The number of messages is not the same as number of rounds. A protocol with 2 rounds and 4 messages can be converted into one with 3 rounds and 3 messages by combining messages.

2.2 Universal composability

The Universal Composability (UC) model, introduced by Canetti [Can01], is an alternative to the game-based model of security more commonly used in cryptography. As the name suggests, protocols proven secure in the UC model can be run in parallel with arbitrarily many copies of itself and other protocols. The model frames cryptographic protocols as idealized *functionalities*, which can be thought of as black boxes with a tightly constrained interface to the outside world. In the security game showing Π *UC-realizes* the functionality $\mathcal{F}$, the goal of an interactive Turing machine called the *environment* $\mathcal{Z}$ is to distinguish between the *ideal world* and the *real world*, which are defined as follows.

In the *real world* all the parties participate in Π, $\mathcal{Z}$ may view parties' outputs, and is permitted to give arbitrary instructions to a separate interactive Turing machine, called the *adversary* $\mathcal{A}$. The environment can arbitrarily ask the adversary to view/modify/delay/drop messages between parties, corrupt parties, and interact with any ideal functionalities $\mathcal{F}_i'$ used to instantiate Π. In the *ideal world* all the parties are *dummies*, speaking directly to $\mathcal{F}$. In addition, the adversary may also only interact to $\mathcal{F}$, though it is still permitted to corrupt parties.

Π UC-realizes $\mathcal{F}$ if for any $\mathcal{A}$, there exists a simulator $\mathcal{S}$ of $\mathcal{A}$ such that any $\mathcal{Z}$ has at most negligible advantage in distinguishing between $\mathcal{A}$ and $\mathcal{S}$. That is,

$$\textsc{Ideal}^{\mathcal{F}}_{\mathcal{S},\mathcal{Z}} \overset{c}{\approx} \textsc{Exec}_{\mathcal{A},\mathcal{Z}}$$

where the LHS refers to the probability ensemble consisting of the view of the environment in the ideal world, and the RHS in the real world.

By [Can01, Theorem 11], it suffices to imagine that the adversary $\mathcal{A}$ is the *dummy adversary* $\mathcal{A}_D$, which simply delivers backdoor messages generated by the environment to the specified recipients, and delivers to the environment all backdoor messages generated by the protocol parties, as well as the sender machine's identity.

There are two models in which the environment can corrupt parties. In the *static corruption model*, the adversary may not corrupt any parties once execution has started. In the *adaptive corruption model*, the adversary may corrupt parties at any time. In this work, we will consider adversaries who are allowed adaptive corruptions.

To represent different instances of the same scheme, the UC model uses *session identifiers.* Each party is given a session identifier sid on activation and will only interact with other parties with the same sid. To represent individual protocol executions, we will use *sub-session identifiers*, denoted ssid (these can be established by out-of-band means, or, e.g., by exchanging nonces [BLR04]). A party in session sid may be engaged in multiple simultaneous protocol executions $\mathsf{ssid}_1, \dots, \mathsf{ssid}_n$. Every protocol execution is uniquely identified by its $(\mathsf{sid}, \mathsf{ssid})$ pair. For clarity of presentation, we will assume in our protocol definitions that session and sub-session identifier establishment has already occurred.

```
Session management
On $(\mathsf{NewSession}, \mathsf{ssid}, \mathcal{P}_j, \mathsf{pw}_i, \mathsf{role})$ from $\mathcal{P}_i$
  send $(\mathsf{NewSession}, \mathsf{ssid}, \mathcal{P}_i, \mathcal{P}_j)$ to $\mathcal{S}$
  if $\nexists(\mathsf{Session}, \mathsf{ssid}, \mathcal{P}_i, \mathcal{P}_j, \mathsf{pw}_i, \mathsf{role})$ :
    record $(\mathsf{Session}, \mathsf{ssid}, \mathcal{P}_i, \mathcal{P}_j, \mathsf{pw}_i, \mathsf{role})$
    Mark it fresh

Active session attack
On $(\mathsf{TestPwd}, \mathsf{ssid}, \mathcal{P}_i, \mathsf{pw}')$ from $\mathcal{S}$
  retrieve $(\mathsf{Session}, \mathsf{ssid}, \mathcal{P}_i, \mathcal{P}_j, \mathsf{pw}_i, \mathsf{role})$
   marked fresh
  if $\mathsf{pw}_i = \mathsf{pw}'$ :
    Mark session compromised
    send "correct" to $\mathcal{S}$
  else
    Mark session interrupted
    send "wrong" to $\mathcal{S}$

Key generation and authentication
On $(\mathsf{NewKey}, \mathsf{ssid}, \mathcal{P}_i, K')$ from $\mathcal{S}$
  retrieve $(\mathsf{Session}, \mathsf{ssid}, \mathcal{P}_i, \mathcal{P}_j, \mathsf{role}, \mathsf{pw}_i)$
   not marked completed
  if session is compromised :
    $K_i \coloneqq K'$
  elseif session is fresh AND
   $\exists(\mathsf{Key}, \mathsf{ssid}, \mathcal{P}_j, \mathsf{pw}_j, K_j)$ s.t. $\mathsf{pw}_i = \mathsf{pw}_j$ :
    $K_i \coloneqq K_j$
  else : $K_i \leftarrow\$ \{0,1\}^\lambda$
  if session is fresh :
    record $(\mathsf{Key}, \mathsf{ssid}, \mathcal{P}_i, \mathsf{pw}_i, K_i)$
  Mark session completed
  send $(\mathsf{ssid}, K_i)$ to $\mathcal{P}_i$
```

Figure 1: The $\mathcal{F}_{\mathsf{PAKE}}$ ideal functionality, with the modifications suggested in [AHH21].

2.3 PAKE and iPAKE

PAKE. We describe the function of a PAKE protocol and its intended security properties. A *(balanced) password authenticated key-exchange protocol* is a two-party key-exchange protocol where parties use mutual knowledge of a low-entropy password pw to establish a high-entropy *session key* K. The security goals for a PAKE are (1) to establish a high-entropy shared session key when both parties are honest, (2) to prevent passive adversaries from learning anything about the password, and (3) to limit active adversaries to one (or another small constant) password guess(es) per protocol instance, even if given access to session keys. We give the corresponding ideal functionality $\mathcal{F}_{\mathsf{PAKE}}$ in Figure 1. We include the small modifications suggested in [AHH21], as do the authors of CHIP/CRISP.

Catastrophic impersonation in PAKE. PAKE protocols permit an attacker who corrupts a party to learn pw and subsequently impersonate *any* party using the same password. In a setting where multiple devices all share the same password, this results in *catastrophic impersonation* following the compromise of even a single device.

iPAKE. *Identity-binding PAKE* (iPAKE) [CNPR22] was introduced to mitigate catastrophic impersonation. Each party runs a one-time initialization procedure $\mathsf{StorePwdFile}$ that takes as input a (common) password pw and its (public) identity id, and outputs a *password file* pwfile. That party then saves pwfile, and deletes pw. At a later point in time, two parties who wish to authenticate run the iPAKE protocol using their respective password files. The output is a tuple (K, id) containing the session key and the identity of the other party.

As with PAKE, iPAKE provides a high-entropy shared key to the two parties running the protocol, and limits active attackers to a single password guess per instance. In contrast to PAKE, however, iPAKE also ensures some measure of robustness following compromise of parties. Specifically, an attacker who compromises Alice only learns Alice's pwfile. With this, the attacker can impersonate Alice to anyone, but does not immediately learn enough information to impersonate any other party to anyone else (including Alice). The best the attacker can do—which is clearly unavoidable in our setting—is to mount a brute-force attack against Alice's pwfile to derive pw; this can be made prohibitive if the pw has moderate entropy and/or if $\mathsf{StorePwdFile}$ is relatively slow. The latter can be achieved by incorporating into $\mathsf{StorePwdFile}$ a time-, memory-, and/or cache-intensive password hashing function, such as Argon2 [BDK15] or bscrypt [Tho22a].

We briefly describe the function of the remaining procedures of $\mathcal{F}_{\mathsf{iPAKE}}$. The formal definition is given in Figure 2.

Corruption. iPAKE permits two kinds of corruption queries:

$\mathsf{StealPwdFile}$ Returns the password file of a specific party $\mathcal{P}$, and does *not* mark $\mathcal{P}$ corrupted, i.e., the adversary does not get any ephemeral keys nor does it gain control over $\mathcal{P}$.[4]

$\mathsf{Corrupt}$ Returns the output of $\mathsf{StealPwdFile}$ in addition to all the ephemeral data held by $\mathcal{P}$, sends a message to the environment that the action was performed on $\mathcal{P}$, and *does* mark $\mathcal{P}$ corrupted.

We say that a party who is not corrupted but whose password file has been stolen is *compromised.*

OfflineTestPwd. This is called by the adversary $\mathcal{A}$ to guess the password of a stolen password file.[5]

OfflineComparePwd. This is called by $\mathcal{A}$ to compare the password files of compromised parties. Through this, it learns which parties' passwords are equal, but does not learn the password itself.

NewSession. This is called by a party $\mathcal{P}_i$ to initiate a new iPAKE session with $\mathcal{P}_j$. At some point, $\mathcal{P}_j$ will have to call $\mathsf{NewSession}$ with $\mathcal{P}_i$ and the same sid and ssid if they wish to complete the protocol execution.

NewKey. This is called by $\mathcal{A}$ with a session key K' and ID id' to finalize the key exchange in an active session. If the session is `fresh`, i.e., has not already been interrupted with a different attack, then this outputs a uniformly random key (independent of K') and actual peer identity to the relevant party, and it

[4]This is not explicitly described as a corruption query in its original formalization [GMR06], but, as Hesse [Hes20] explains, this is the only definitionally sound way to treat this query.

[5]In order to prevent trivial simulators with arbitrary brute-force attack capability, we bound the simulator's usage of $\mathsf{OfflineTestPwd}$ by the runtime of the environment $\mathcal{Z}$. See Hesse [Hes20] for more detail.

Password Registration

```
On $(\mathsf{StorePwdFile}, \mathsf{sid}, \mathsf{id}, \mathsf{pw})$ from $\mathcal{P}$
  if $\nexists$ a record $(\mathsf{File}, \mathcal{P}, \mathsf{id}, \mathsf{pw})$ :
    record $(\mathsf{File}, \mathcal{P}, \mathsf{id}, \mathsf{pw})$
```

Password Authentication

```
On $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{ssid}, \mathcal{P}_j)$ from $\mathcal{P}_i$
  if $\exists$ a record $(\mathsf{Session}, \mathsf{sid}, \mathsf{ssid}, \mathcal{P}_j, \mathcal{P}_i, \cdot)$ :
  retrieve $(\mathsf{File}, \mathcal{P}_i, \mathsf{id}_i, \mathsf{pw}_i)$
  send $(\mathsf{NewSession}, \mathsf{ssid}, \mathcal{P}_i, \mathcal{P}_j, \mathsf{id}_i)$ to $\mathcal{A}$
  if $\nexists$ a record $(\mathsf{Session}, \mathsf{sid}, \mathsf{ssid}, \cdot, \cdot, \cdot)$ :
    record $(\mathsf{Session}, \mathsf{sid}, \mathsf{ssid}, \mathcal{P}_i, \mathcal{P}_j, \mathsf{pw}_i)$
```

Active Session Attacks

```
On $(\mathsf{OnlineTestPwd}, \mathsf{sid}, \mathsf{ssid}, \mathcal{P}_i, \mathsf{pw}')$ from $\mathcal{A}$
  retrieve $(\mathsf{Session}, \mathsf{sid}, \mathsf{ssid}, \mathcal{P}_i, \cdot, \mathsf{pw}_i)$
   marked fresh or compromised
  if $\mathsf{pw}' = \mathsf{pw}_i$ :
    record $(\mathsf{Imp}, \mathsf{ssid}, \mathcal{P}_i, *)$
    Mark the session compromised
    send "correct" to $\mathcal{A}$
  else :
    Mark the session interrupted
    send "wrong" to $\mathcal{A}$

On $(\mathsf{Impersonate}, \mathsf{sid}, \mathsf{ssid}, \mathcal{P}_i, \mathcal{P}_k)$ from $\mathcal{A}$
  retrieve $(\mathsf{Session}, \mathsf{sid}, \mathsf{ssid}, \mathcal{P}_i, \cdot, \mathsf{pw}_i)$
   marked fresh or compromised
  retrieve $(\mathsf{File}, \mathcal{P}_k, \mathsf{id}_k, \mathsf{pw}_k)$
   marked compromised
  if $\mathsf{pw}_i = \mathsf{pw}_k$ :
    record $(\mathsf{Imp}, \mathsf{ssid}, \mathcal{P}_i, \mathsf{id}_k)$
    Mark the session compromised
    send "correct" to $\mathcal{A}$
  else :
    Mark the session interrupted
    send "wrong" to $\mathcal{A}$
```

Key Generation and Authentication

```
On $(\mathsf{NewKey}, \mathsf{sid}, \mathsf{ssid}, \mathcal{P}_i, \mathsf{id}', K')$ from $\mathcal{A}$
  retrieve $(\mathsf{Session}, \mathsf{sid}, \mathsf{ssid}, \mathcal{P}_i, \mathcal{P}_j, \mathsf{pw}_i)$
   not marked completed
  retrieve $(\mathsf{File}, P_j, \mathsf{id}_j, \mathsf{pw}_j)$
  assert $\neg$(session is fresh and $\mathsf{id}' \neq \mathsf{id}_j$) and
   $\neg$(session is compromised and $(\mathsf{Imp}, \mathsf{ssid}, \mathcal{P}_i, i)$
     is not recorded for both $i \in \{\mathsf{id}', *\}$)
  if session is compromised : $K_i \coloneqq K'$
  elif session is fresh and $\exists$ a record
   $(\mathsf{Key}, \mathsf{ssid}, \mathcal{P}_j, \mathsf{pw}_j, K_j)$ with $\mathsf{pw}_i = \mathsf{pw}_j$ :
    $K_i \coloneqq K_j$
  else : $K_i \leftarrow\!\$\ \{0,1\}^\lambda$
  if session is fresh :
    record $(\mathsf{Key}, \mathsf{ssid}, \mathcal{P}_i, \mathsf{pw}_i, K_i)$
  Mark session completed
  send $(\mathsf{Key}, \mathsf{ssid}, \mathsf{id}', K_i)$ to $\mathcal{P}_i$
```

Stealing Password Data

```
On corruption query $(\mathsf{StealPwdFile}, \mathcal{P})$ from $\mathcal{A}$:
  retrieve $(\mathsf{File}, \mathcal{P}, \mathsf{id}, \mathsf{pw})$
  Mark the file compromised
  if $\exists$ a record $(\mathsf{Offline}, \mathcal{P}, \mathsf{pw})$ :
    send $(\mathsf{Stolen}, \mathsf{id}, \mathsf{pw})$ to $\mathcal{A}$
  else : send $(\mathsf{Stolen}, \mathsf{id}, \perp)$ to $\mathcal{A}$

On $(\mathsf{OfflineTestPwd}, \mathcal{P}, \mathsf{pw}')$ from $\mathcal{A}$
  retrieve $(\mathsf{File}, \mathcal{P}, \cdot, \mathsf{pw})$
  if file marked compromised :
    if $\mathsf{pw}' = \mathsf{pw}$ : send "correct" to $\mathcal{A}$
    else : send "wrong" to $\mathcal{A}$
  else : record $(\mathsf{Offline}, \mathcal{P}, \mathsf{pw}')$

On $(\mathsf{OfflineComparePwd}, \mathsf{sid}, \mathcal{P}_i, \mathcal{P}_j)$ from $\mathcal{A}$
  retrieve $(\mathsf{File}, \mathcal{P}_i, \cdot, \mathsf{pw}_i)$ is compromised
  retrieve $(\mathsf{File}, \mathcal{P}_j, \cdot, \mathsf{pw}_j)$ is compromised
  if $\mathsf{pw}_i = \mathsf{pw}_j$ : send "match" to $\mathcal{A}$
  else : send "no match" to $\mathcal{A}$
```

```
On $(\mathsf{OnlineComparePwd}, \mathsf{sid}, \mathsf{ssid}, \mathcal{P}_i, \mathcal{P}_j)$ from $\mathcal{A}$
  retrieve $(\mathsf{Session}, \mathsf{sid}, \mathsf{ssid}, \mathcal{P}_i, \cdot, \mathsf{pw}_i)$
   marked fresh or compromised
  retrieve $(\mathsf{File}, \mathcal{P}_j, \cdot, \mathsf{pw}_j)$ marked compromised
  if $\mathsf{pw}_i = \mathsf{pw}_j$ : send "match" to $\mathcal{A}$
  else :
    Mark the session interrupted
    send "no match" to $\mathcal{A}$
```

Figure 2: The $\mathcal{F}_{\mathsf{iPAKE}}$ and $\mathcal{F}^{\mathrm{ocw}}_{\mathsf{iPAKE}}$ ideal functionalities. Everything outside the dashed box belongs to $\mathcal{F}_{\mathsf{iPAKE}}$. Everything including the dashed box belongs to $\mathcal{F}^{\mathrm{ocw}}_{\mathsf{iPAKE}}$.

will output the same key and actual peer identity to the other party when called for them. The values K' and id' are sent to the relevant party when certain compromise conditions are met. These are described below.

OnlineTestPwd. This is called by the $\mathcal{A}$ to guess the password for an active session. On success, the session is marked `compromised`, meaning the adversary has full control over the session key and perceived identity at both parties. On failure, the session is marked `interrupted`, meaning the session keys will be random and distinct, i.e., the exchange will fail.

Impersonate. This is called by $\mathcal{A}$ to impersonate a compromised user in an active session. If the compromised user's pw matches the session peer's pw, then the adversary is given full control over the peer's session key (*not* the impersonated user's session key), and the identity is set to the impersonated party's identity.

2.3.1 Adding OnlineComparePwd

In order to permit analysis of LATKE, we must slightly extend the $\mathcal{F}_{\mathsf{iPAKE}}$ functionality. We add one procedure, $\mathsf{OnlineComparePwd}$. This procedure allows the adversary to check if a compromised user's password matches the password used in an active session. We call the new functionality $\mathcal{F}^{\mathrm{ocw}}_{\mathsf{iPAKE}}$.

We argue that this additional procedure is natural in the sense that similar attacks are already allowed in other PAKE extensions. The Ω-method aPAKE [GMR06] begins with a PAKE over $\mathsf{H}(\mathsf{pw})$, and requires the client to store pw and the server to store $\mathsf{H}(\mathsf{pw})$. Thus, an active attacker who compromises a server can use the stolen hash to intercede in unrelated PAKE sessions and thus determine whether or not the parties of a given session share a password. The reason an $\mathsf{OnlineComparePwd}$ procedure is not included in aPAKE is because aPAKE doesn't a strong notion of identity—to perform the described attack, it suffices to use $\mathsf{Impersonate}$ using the generic identities of "client" or "server". This isn't so in iPAKE: an $\mathsf{Impersonate}$ query would force the attacker to use the unique identity of the compromised user.

The procedure is natural in another sense. It is possible to emulate $\mathsf{OnlineComparePwd}$ by calling $\mathsf{Impersonate}$ as described above, tearing down the session with $\mathsf{OnlineTestPwd}(\mathsf{sid}, \mathsf{ssid}, \mathcal{P}, \perp)$, and forcing the parties to attempt a new session. The only step that is outside of the UC simulator's capabilities is the last. In reality, though, this is already possible. For example, WPA2-protected WiFi connections can be injected with a deauthentication packet, thus forcing the client to tear down the session and reconnect [SRV23]. Similar attacks in other protocols such as Bluetooth are also documented [Lou21]. Thus, it is not unreasonable to imagine ideal attacker being able to force a retry in just the key exchange portion of a protocol.

2.4 Key exchange

We review notions of authenticated key exchange (AKE), and identity-based key exchange (IBKE). We will include definitions for *post-specified peer* protocols, i.e., protocols wherein the parties do not know the identity of their interlocutor in advance.

2.4.1 Authenticated key exchange

An authenticated key exchange protocol (AKE) is a cryptographic protocol wherein two parties interact over a public channel to produce a shared secret key. The procedures of a post-specified peer AKE are as follows:

$\mathsf{KeyGen}(1^\lambda) \to (\mathsf{upk}, \mathsf{usk})$ Generates a long-term keypair.

$\mathsf{Execute}(\mathsf{sk}_A) \Leftrightarrow \mathsf{Execute}(\mathsf{sk}_B)$ Executes the interactive key exchange protocol. The output is a shared session key K and the public key of the other party, pk_A or pk_B.

Identity-based key exchange. Identity-based key exchange (IBKE) is a generalization of AKE which introduces a trusted third party, called the *key generation center* (KGC). The KGC is responsible for generating the *main keypair* $(\mathsf{mpk}, \mathsf{msk})$, and for *extracting* secret keys for users. A user with an identifier string id will request a secret key $\mathsf{sk}_{\mathsf{id}}$ from the KGC that corresponds to id. With this secret key, the user can participate in key establishment protocols wherein the other participant knows only the global mpk and their id (and optionally their own $\mathsf{sk}_{\mathsf{id}'}$ if mutual authentication is desired).

Concretely, the procedures of a post-specified peer IBKE are as follows:

$\mathsf{Setup}(1^\lambda) \to (\mathsf{mpk}, \mathsf{msk})$ Generates the main keypair.

$\mathsf{Extract}_{\mathsf{msk}}(\mathsf{mpk}, \mathsf{id}, \mathsf{aux}) \to \mathsf{sk}_{\mathsf{id}}$ Extracts a secret key for the given ID under the given mpk, with auxiliary data aux.

$\mathsf{Execute}(\mathsf{sk}_A, \mathsf{id}_A, \mathsf{mpk}) \Leftrightarrow \mathsf{Execute}(\mathsf{sk}_B, \mathsf{id}_B, \mathsf{mpk})$ Executes the interactive key exchange protocol. The output, on success, is the shared session key K and the ID of the other party, id_A or id_B.

Identity concealment. A post-specified peer AKE or IBKE is said to have *passive responder concealment* if the identity is hidden from a passive adversary, and *active responder concealment* if from an active adversary (i.e., a malicious initiator). Specifically, in a protocol with active responder concealment, the responder will wait until the initiator has authenticated themselves before they send any identifying information. Active initiator concealment is defined similarly. A protocol may have both passive initiator concealment and active responder concealment (as SIGMA-R does; Section 4) or vice-versa, but cannot have active concealment for both parties, since one party must authenticate first.

2.4.2 Canetti-Krawczyk (CK) model

The Canetti-Krawczyk (CK) model [CK01] and its extension, the identity-based CK model (id-CK) are common models for proving security of AKEs and IBKEs, respectively. Their purpose is to model everything an adversary can reasonably do in a real-world AKE or IBKE protocol execution. Within these models, it is possible to define notions of ordinary *session-key (SK) security*, key-compromise impersonation resistance (KCIR), maximal exposure resistance (MEX), forward secrecy (FS), post-compromise security (PCS), and more. In order to capture identity concealment, we will use the CK model adapted to the post-specified peer setting, as presented in [CK02a].

For the security of LATKE, we will require an IBKE with SK security with KCIR and *full FS*. In words, KCIR ensures that, if Mallory compromises Alice, Mallory cannot impersonate anyone to Alice (aside from the other parties Mallory has already compromised). FS ensures that if Mallory compromises Alice, Mallory remains unable to learn the session keys from Alice's past completed sessions. IBKE has another notion of FS, which we call *KGC-FS*, which requires that, if Mallory compromises the KGC, she is still unable to learn the session keys of Alice's past completed sessions. We say a protocol has *full FS* if it has FS and KGC-FS. More detailed definitions can be found in Appendix A.3.

Finally, we note there are numerous, subtly different, variants of the CK model, including $\mathrm{CK}_{\mathrm{HMQV}}$, eCK, and CK^+ [Kra05, LLM07, FSXY12] (their id- variants being defined similarly, by adding KGC extraction and revelation). In fact, it has been shown that security in the first three variants is pairwise incomparable—security in one model does not imply security in any other model [Cre11]. Nevertheless, these models have enough in common that our main security reduction will be able to use any one of them.

2.5 Symmetric encryption

LATKE requires a symmetric encryption scheme that is secure under adaptive corruptions. For this, we will use SIM*-AC-CCA encryption, introduced by Jäger and Tyagi [JT20, Jae23].

We choose this security notion over, e.g., CCA2 security, for several practical reasons. For our UC proof to go through, encryption needs to be *non-committing*—a simulator must be able to open a ciphertext to whichever value it chooses, so long as the adversary does not already know the key. This is a common enough requirement that many UC protocols create their own non-committing encryption scheme from a (programmable) random oracle or an ideal cipher. These constructions have two problems: (1) they are not generalizable, i.e., it is not clear which other authenticated encryption schemes can be plugged into the protocol, and (2) the proofs of security of these bespoke constructions often have subtle bugs (as demonstrated by Jäger and Tyagi). Thus, we opt for a generic notion of simulatable authenticated encryption.

SIM*-AC-CCA security. Informally, an authenticated encryption scheme AE is SIM*-AC-CCA-secure with respect to some ideal primitive P (e.g., programmable random oracle or ideal cipher), if there is a simulator $\mathsf{S}_{\mathsf{cca}}$ such that no PPT adversary can distinguish between $\mathsf{S}_{\mathsf{cca}}$ and AE, even when given the ability to encrypt messages, decrypt messages, expose keys, and program the random oracle. We refer the reader to [Jae23] for the formal definition of security. For this work, it will suffice to list the specific operations available to the symmetric encryption scheme, AE, and the associated simulator $\mathsf{S}_{\mathsf{cca}}$.

The procedures exposed by the symmetric encryption scheme AE are the usual ones: KeyGen for key generation, $\mathsf{Enc}^{\mathsf{P}}$ for encryption (with access to primitive P), and $\mathsf{Dec}^{\mathsf{P}}$ for decryption (also with access to P). The associated simulator $\mathsf{S}_{\mathsf{cca}}$ exposes the following procedures. For brevity, we omit details about initialization of ideal primitives.

$\mathsf{Dec}(u, c) \to m$ Simulates a party u decrypting a ciphertext c.

$\mathsf{Exp}(u, M_u) \to k$ Simulates the corruption, i.e., exposure of a key, of a party u. M_u is the set of ciphertext-plaintext pairs that the simulator must behave consistently with.

Since each step of each subsessions has a unique key, we will let a user u represent a tuple containing a party identifier, all subsession information, and the current step in the protocol execution $(\mathcal{P}, \mathsf{sid}, \mathsf{ssid}, \mathsf{step})$. When $\mathcal{P}$ is corrupted, it will trigger the exposure of all its incomplete subsessions.

Concrete instantiation. We can instantiate a SIM*-AC-CCA-secure encryption scheme is via the encrypt-then-MAC (EtM) transform [JT20]. If SE is SIM*-AC-CPA-secure with primitive P_1 and MAC is UF-CMA-secure with primitive P_2, then $\mathsf{EtM}[\mathsf{SE}, \mathsf{MAC}]$ is SIM*-AC-CCA-secure with primitive $\mathsf{P}_1 \times \mathsf{P}_2$.

If AES is modeled as an ideal cipher, then AES-CTR is SIM*-AC-CPA-secure with primitive $\mathcal{F}_{\mathsf{IC}}$. And if the hash function H is modeled as a random oracle, then $\mathsf{HMAC}[\mathsf{H}]$ is UF-CMA-secure with primitive $\mathcal{F}_{\mathsf{RO}}$. Thus, AES-CTR + HMAC is SIM*-AC-CCA-secure with primitive $\mathcal{F}_{\mathsf{IC}} \times \mathcal{F}_{\mathsf{RO}}$ [JT20].

Similarly, if the ChaCha20 block function [Ber08] is modeled as a random oracle, then it achieves SIM*-AC-PRF security with primitive $\mathcal{F}_{\mathsf{RO}}$. This makes the ChaCha20 stream cipher a SIM*-AC-CPA-secure encryption scheme that can also be plugged into EtM.

3 Construction

In this section we present the LATKE framework for constructing iPAKE protocols. The structure of LATKE can be viewed as a synthesis of an early aPAKE known as the Ω-*method* [GMR06], with the CHIP iPAKE [CNPR22]. We introduce both of these constructions and then present ours.

3.1 Ω-method

Recall the purpose of an aPAKE is to permit a client who has a password pw to establish a secure session with a server who holds a one-way function of the password $\mathsf{H}(\mathsf{pw})$. The Ω-method aPAKE, introduced by Gentry, MacKenzie, and Ramzan [GMR06], is a generic construction that relies only on an ordinary PAKE, symmetric-key encryption, and digital signatures. The protocol can be found in Figure 3.

```
$\Omega$.StorePwdFile(sid, pw) $\to$ pwfile$_{sid}$
1 : $(\mathsf{pk}, \mathsf{sk}) \leftarrow \Sigma.\mathsf{Kg}(1^\lambda)$
2 : $(h_0, h_1) := (\mathsf{H}_0(\mathsf{sid}, \mathsf{pw}), \mathsf{H}_1(\mathsf{sid}, \mathsf{pw}))$
3 : $c \leftarrow \mathsf{AE.Enc}_{h_1}(\mathsf{sk})$
4 : $\mathsf{pwfile} := (\mathsf{pk}, h_0, c)$
5 : delete pw, sk
6 : return pwfile$_{sid}$
```

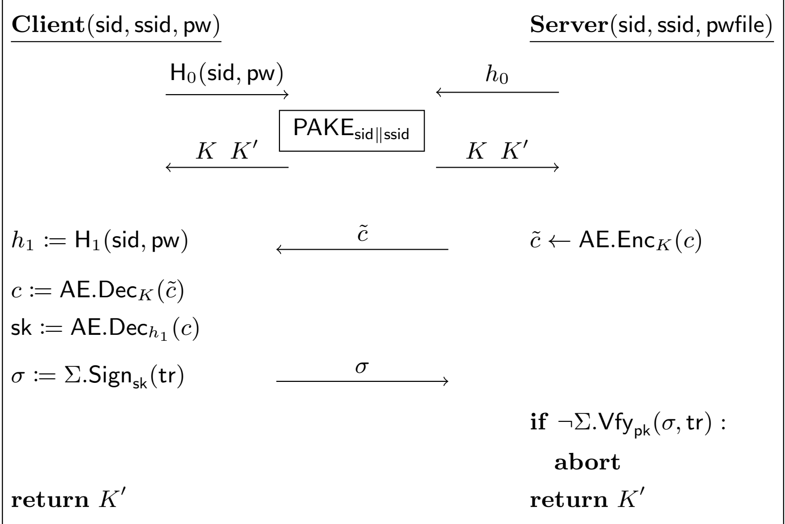

Client(sid, ssid, pw)
Server(sid, ssid, pwfile)
$\mathsf{H}_0(\mathsf{sid}, \mathsf{pw})$
h_0
$\mathsf{PAKE}_{\mathsf{sid}\|\mathsf{ssid}}$
$K\ K'$
$K\ K'$
$h_1 := \mathsf{H}_1(\mathsf{sid}, \mathsf{pw})$
$\tilde{c}$
$\tilde{c} \leftarrow \mathsf{AE.Enc}_K(c)$
$c := \mathsf{AE.Dec}_K(\tilde{c})$
$\mathsf{sk} := \mathsf{AE.Dec}_{h_1}(c)$
$\sigma := \Sigma.\mathsf{Sign}_{\mathsf{sk}}(\mathsf{tr})$
σ
if $\neg\Sigma.\mathsf{Vfy}_{\mathsf{pk}}(\sigma, \mathsf{tr})$:
abort
return K'
return K'

Figure 3: The Ω-method aPAKE [GMR06]. tr represents the protocol transcript up to that point in time. Σ is a signature scheme. AE is an authenticated encryption scheme.

Password file generation. A server with password pw first computes a password identifier $h_0 := \mathsf{H}_0(\mathsf{sid}, \mathsf{pw})$. Next, it computes a signing keypair $(\mathsf{pk}, \mathsf{sk})$ and encrypts sk under the key $h_1 := \mathsf{H}_1(\mathsf{sid}, \mathsf{pw})$ to get ciphertext c. It saves pk, h_0, and the c, and erases pw and sk. To frustrate brute-force attacks, H_0 and H_1 should be hard hash functions.

Key establishment. Both parties perform a fresh PAKE on $\mathsf{H}_0(\mathsf{sid}, \mathsf{pw})$ to derive two keys (K, K'). The parties use K to establish an encrypted channel using the authenticated encryption scheme AE. In that channel, the server sends c. Since the client knows pw, it can decrypt c to sk and use it to compute a signature σ over the entire protocol transcript. The client sends σ to the server, which verifies the signature. On success, both parties output K'.

Key feature: Secure channel. Often, aPAKE constructions are highly bespoke, tailoring themselves to the specific cryptographic assumptions they target. Part of the reason this is true is because they operate under the strict constraint of having a message flow that is statistically independent of the password (otherwise, a passive adversary can collect transcripts and launch a brute-force attack).

The Ω-method construction differs from this. Rather than relying on specific algebraic methods to hide the password, it instead delegates this to a PAKE over the hashed password, which is performed at the beginning of the protocol, and uses the resulting key to set up a *secure channel*. The effect is twofold. Firstly, this excludes all passive adversaries from the secure channel, since the PAKE is secure against passive adversaries. Secondly, this permits the contents of the secure channel *to leak information about the hashed password*. Here, information about the hashed password is leaked because there's an authenticated ciphertext c encrypted under $\mathsf{H}_1(\mathsf{sid}, \mathsf{pw})$. This is permitted because any adversary who can see the messages in this secure channel already knows $\mathsf{H}_0(\mathsf{sid}, \mathsf{pw})$ (since, to see the messages, it must have successfully attacked the PAKE). Thus, if the adversary has gotten so far as to break the secure channel, observing the messages therein doesn't help its brute-force efforts, so long as the hash function H_1 is at least as hard as H_0.

Here, the protocol within the secure channel is extremely simple—it is just one message—but there is nothing preventing this principle from being applied to more complicated protocols which leak information about the hashed password. This will be essential to the flexibility of our iPAKEs.

3.2 CHIP

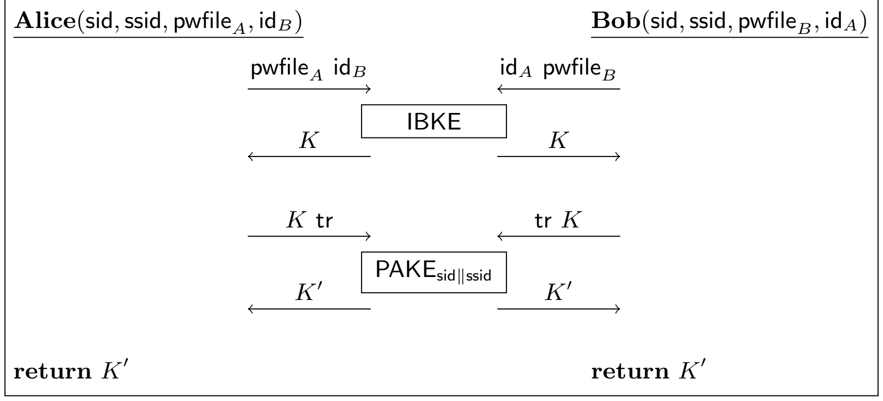
We now look at the first iPAKE construction, CHIP, due to Cremers, Naor, Paz, and Ronen [CNPR22]. Recall the purpose of an iPAKE is to permit parties with mutual knowledge of a password to establish a secure connection while identifying themselves to each other with unique identity strings. Further, if an attacker compromises a device, they should not have the ability to impersonate any identity other than that of the compromised devices (barring a successful brute-force attack).

We give an overview of CHIP in Figure 4. Like the Ω-method, it uses a generic PAKE as a building block. The other building block it requires is any msk-independent identity-based key exchange (IBKE) with KCIR. More formally, the messages of the IBKE must be statistically independent of not just msk, but also the random coins used in the IBKE's Setup procedure.

Password file generation. On initialization, a party with ID string id and password pw hashes pw and uses the output as random coins to generate a fresh IBKE keypair $(\mathsf{msk}, \mathsf{mpk})$. The party then extracts a secret key usk bound to id. The party deletes pw and stores $(\mathsf{id}, \mathsf{mpk}, \mathsf{usk})$.

Key establishment. The CHIP protocol runs in two phases. First, the parties execute the IBKE protocol using their extracted secrets. Note their mpk is guaranteed to be the same if they used the same password. Next, they use the resulting key (plus the transcript of the protocol) as the "password" for an

```
CHIP.StorePwdFile(sid, pw, id) → pwfile
1 :  $(\mathsf{mpk}, \mathsf{msk}) := \mathsf{IBKE.Setup}(1^\lambda; \mathsf{H}_0(\mathsf{sid}, \mathsf{pw}))$
2 :  $\mathsf{usk} \leftarrow \mathsf{IBKE.Extract}_{\mathsf{msk}}(\mathsf{mpk}, \mathsf{id})$
3 :  $\mathsf{pwfile} := (\mathsf{id}, \mathsf{mpk}, \mathsf{usk})$
4 :  delete $\mathsf{pw}, r, \mathsf{msk}$
5 :  return $\mathsf{pwfile}$
```



Alice(sid, ssid, pwfile$_A$, id$_B$)
Bob(sid, ssid, pwfile$_B$, id$_A$)
pwfile$_A$ id$_B$
id$_A$ pwfile$_B$
IBKE
K
K
K tr
tr K
PAKE$_{\mathsf{sid}\|\mathsf{ssid}}$
K'
K'
return K'
return K'

Figure 4: The CHIP iPAKE, using a generic IBKE and PAKE. tr represents the protocol transcript up to that point.

ordinary PAKE. The output is their final session key.

Key feature: The self-KGC trick. The design choice that stands out most in CHIP is that it uses IBKE, a cryptographic primitive which typically requires a trusted third party. The purpose of the IBKE in this protocol is to establish a key that simultaneously binds the ID and the password, while ensuring that (1) the pwfile is bound to a specific ID, and (2) the password is not directly stored. An IBKE is the natural solution to a key exchange with property (1). And to bind the password to the session, it suffices to use the (hashed) password to generate the main long-term key, constrain it to the id, and then discard the intermediate values.

This trick is generic, requiring no specific property of the IBKE other than the most basic notion of security, i.e., an adversary without the password cannot produce msk by accident (though CHIP still needs other properties for the rest of the construction to be secure). We will use this trick as the main identity mechanism in LATKE.

Design limitations. As mentioned in Section 1, the construction of CHIP puts limitations on which IBKEs can be used. Like many PAKE extensions, CHIP's message flow must be independent of the password (and hence the random coins of Setup). This is because, unlike the Ω-method, CHIP's key exchange is done *in the clear*—it must be secure against passive adversaries looking to perform an offline brute-force attack. The narrow choice of IBKE means that CHIP has no plausible post-quantum instantiation. In addition, CHIP has no plausible instantiation with an IBKE with identity concealment.

3.3 LATKE

LATKE combines the key features of CHIP and the Ω-method. The password file generation procedure is identical to that of CHIP, using the self-KGC trick to generate keys that are bound to the password and an identity string. The protocol is similar in structure to the Ω-method, beginning with a PAKE over a collision-resistant one-way function of the password, specifically mpk.[6] The resulting key is used to set up a secure channel for the rest of the protocol. The parties execute an IBKE protocol inside the secure channel and output the result. The full definition of LATKE can be found in Figure 5. The pre- and post-specified peer variants only differ in what is included in the initial PAKE step.

Since the framework permits a wider range of IBKEs than CHIP, we can now achieve post-quantum iPAKE from post-quantum IBKE and PAKE, and identity-concealing iPAKE from identity-concealing IBKE.

There are a handful of details to work out in order to determine the security and efficiency of LATKE.

Secure channel: The EUE transform. We must specify what we mean by *secure channel.* In the Ω-method, the payload of the channel was a single message. In the case of LATKE, the payload is the entire transcript of an IBKE. In order to achieve security under adaptive corruptions, we must pick our notion of authenticated encryption carefully, and design a secure channel protocol that can be simulated even for sessions between parties whose password files are unknown to the UC simulator. We will design a simple transform, called *encrypt-and-unconditionally execute* (EUE) to meet this specification.

Necessary security properties. Another detail is precisely what kind of security we need from the PAKE and IBKE. The PAKE requirement does not change from the Ω-method—all we require is a protocol that UC-realizes $\mathcal{F}_{\mathsf{PAKE}}$. The IBKE must have KCIR and full FS, which we note are common properties of key exchange protocols. In the lead-up to our theorem statement, we will explain why each property is necessary.

Building the IBKE from AKE. To further demonstrate the practicality of LATKE, we will show that it is possible to generically achieve these properties given just an AKE with KCIR (and no FS).

Round complexity. PAKE and IBKE constructions can both be achieved using one round of communication each. So, like CHIP, the minimum round

[6]It would suffice to perform the PAKE over a hard hash of pw rather than mpk. However, since parties already store mpk, and because mpk values do not collide (otherwise the IBKE would be trivially insecure), it is just as good to use mpk.

```
LATKE.StorePwdFile(sid, pw, id)
1 : $(\mathsf{mpk}, \mathsf{msk}) := \mathsf{IBKE.Setup}(1^\lambda; \mathsf{H}_0(\mathsf{sid}, \mathsf{pw}))$
2 : $\mathsf{usk} \leftarrow \mathsf{IBKE.Extract}_{\mathsf{msk}}(\mathsf{mpk}, \mathsf{id})$
3 : $\mathsf{pwfile} := (\mathsf{id}, \mathsf{mpk}, \mathsf{usk})$
4 : delete $\mathsf{pw}, r, \mathsf{msk}$
5 : return $\mathsf{pwfile}$
```

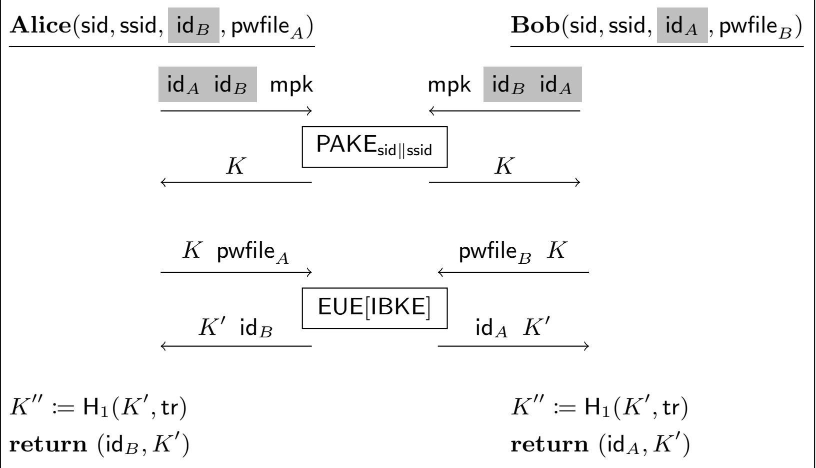


Figure 5: The LATKE iPAKE framework. With the grey text excluded, the above defines LATKE$^{\text{post}}$. With the grey text included, the above defines LATKE$^{\text{pre}}$. EUE[IBKE] is encrypt-and-unconditionally-execute transform of the identity-based key agreement protocol IBKE (Section 3.3.1). tr is the transcript of the entire protocol.

complexity for the LATKE framework is two rounds. While one-round PAKEs and IBKEs exist (e.g., EKE [LLHG23] and Fiore-Gennaro [FG10], resp.), most rely on a Diffie-Hellman-type assumption. In Section 5, we discuss ideas to achieve post-quantum LATKE in two rounds.

3.3.1 Encrypt-and-unconditionally-execute transform

Recall we must use the PAKE-derived key to establish a secure channel in which to perform the IBKE protocol. To hide IBKE messages and ensure correct execution, we must carefully apply a symmetric encryption scheme so as to avoid potential replay attacks. In addition, our security proof requires that the protocol transcript look the same to a passive attacker regardless of whether the initial PAKE succeeded. To achieve this, we must send ciphertexts *even on protocol failure*. Finally, since we must be able to open ciphertexts to arbitrary values, we require a simulation-secure notion of symmetric encryption. We call the combination of these techniques the *encrypt-and-unconditionally-execute transform* $\mathsf{EUE}[\mathsf{IBKE}]$.

EUE definition. We require an IBKE with a fixed number of rounds and fixed message sizes, an authenticated encryption scheme AE with SIM*-AC-CCA security using ideal primitive P, and a random oracle H. We use the variable rmode to denote whether the party is in *real mode*, i.e., is still encrypting real messages. At the beginning of the protocol, $\mathsf{rmode} \coloneqq \mathtt{true}$. We define the behavior of a party $\mathcal{P}$ interacting with its peer $\mathcal{P}'$ in session $(\mathsf{sid}, \mathsf{ssid})$. We let K represent the encryption key that $\mathcal{P}$ intends to use for the session.

1. Let the *chain key* $K_{\mathsf{ch}} \coloneqq \mathsf{H}(\mathsf{sid}, \mathsf{ssid}, 0 \| K)$. We will use the chain key to derive all subsequent keys using a symmetric ratchet similar to the Signal protocol [Mar16, CJSV22]. Once a chain key has been ratcheted forward, the old keys are erased.

2. Protocol step i, where $\mathcal{P}$ is sending: Compute the step key and new chain key $(K_{\mathsf{ch}}, k_i) \coloneqq \mathsf{H}(\mathsf{sid}, \mathsf{ssid}, 1 \| K_{\mathsf{ch}})$. If $\mathsf{rmode} = \mathtt{true}$, the IBKE hasn't aborted yet, so $\mathcal{P}$ has a specific IBKE message m_i it wishes to send. Compute $c_i \coloneqq \mathsf{AE.Enc}^{\mathsf{P}}_{k_i}(m)$. Otherwise if $\mathsf{rmode} = \mathtt{false}$, let $c_i \coloneqq \mathsf{AE.Enc}_{k_i}(0^{\ell_i})$, where 0^{ℓ_i} is the all-zeros string of the known step i message length ℓ_i. Finally send c_i to $\mathcal{P}'$.

3. Protocol step i, where $\mathcal{P}$ is receiving: Compute the step key as described above. If $\mathsf{rmode} = \mathtt{false}$, the IBKE has aborted, so simply return. Otherwise, proceed as follows. To decrypt the incoming ciphertext c_i, compute $m'_i \coloneqq \mathsf{AE.Dec}^{\mathsf{P}}_{k_i}(c_i)$. If a decryption failure occurred, set $\mathsf{rmode} \coloneqq \mathtt{false}$ and return. Otherwise pass m'_i to $\mathcal{P}$ to continue the protocol execution. If $\mathcal{P}$ aborts, then abort the protocol.

4. At the end of the protocol, if $\mathsf{rmode} = \mathtt{false}$, output $\perp$. Otherwise, output $\mathcal{P}$'s output.

We will not prove any standalone security properties about the EUE protocol, as it only makes sense when used in the broader context of LATKE.

3.3.2 Building the IBKE

We briefly recall results which make it possible to construct a satisfactory IBKE from the smaller building block of authenticated key exchange (AKE). We start with an AKE with just key compromise impersonation resistance (KCIR). We will use these techniques to build several IBKEs which we benchmark in Section 4.

Forward secrecy for AKE. LATKE requires full forward secrecy from its key exchange. There are a few well known methods to endow a generic AKE with forward secrecy.

For generic AKEs, Boyd and Nieto [BN11] describe a transform wherein the AKE is used to set up an authenticated channel, and with that channel the parties perform an ordinary (unauthenticated) key exchange to derive the final key. The resulting AKE has forward secrecy as long as the adversary is not given the ability to perform ephemeral key revelations. This security meets the preconditions of Theorem 1.

Another transform exists for AKEs with *weak forward secrecy*, i.e., FS when the adversary does not tamper with the messages in the test session. For these protocols it suffices to simply add *key confirmation*—each party sends a MAC whose key is derived from the shared secret [Kra05, Section 8] [GGJJ23].

Both of these transforms add at least one round of communication.

AKE to IBKE. To convert an AKE into an identity-based AKE, we follow a well-known schema, the *certification approach*, used for building identity-based signatures from ordinary signatures [KN09]. In this construction, every user has an AKE public key pk, and some ID string id. The key generation center (KGC) issues identities by signing $(\mathsf{id}, \mathsf{pk})$ pairs with its signing key msk. The user's *certificate* is thus $(\mathsf{id}, \mathsf{pk}, \sigma)$, where σ is the signature they received from the KGC. At the very beginning of key exchange, users exchange and verify each other's certificates with respect to the KGC's public key mpk. On success, they proceed with AKE using the user-provided public keys.

We describe the transform more formally. Let Σ be an EUF-CMA-secure signature scheme, and let AKE be an authenticated key exchange protocol in the post-specified peer model. The new post-specified peer IBKE is defined as follows:

$\mathsf{Setup}(1^\lambda) \to (\mathsf{mpk}, \mathsf{msk})$ Generates a signing keypair $(\mathsf{mpk}, \mathsf{msk}) \leftarrow \Sigma.\mathsf{KeyGen}(1^\lambda)$.

$\mathsf{Extract}_{\mathsf{msk}}(\mathsf{id}, \mathsf{upk}) \to \sigma$ Computes the signature $\sigma \leftarrow \Sigma.\mathsf{Sign}_{\mathsf{msk}}(\mathsf{id} \| \mathsf{upk})$.

$\mathsf{Execute}(\mathsf{usk}_A, \mathsf{id}_A, \sigma_A, \mathsf{mpk}) \Leftrightarrow \mathsf{Execute}(\mathsf{usk}_B, \mathsf{id}_B, \sigma_B, \mathsf{mpk})$ Each party first sends each other $\mathsf{cert}_X \coloneqq (\mathsf{upk}_X, \mathsf{id}_X, \sigma_X)$ where $X \in \{A, B\}$. Then, each party verifies cert with respect to mpk. On success, the parties proceed with the AKE. At the end, the parties check that the outputted upk matches the one they received.

For a pre-specified peer IBKE from a pre-specified peer AKE, it suffices to remove the last check and add a check that the received certificate's ID matches the expected ID.

We make some remarks on the certification approach. Firstly, the security properties we desire, namely KCIR and FS, are preserved in this transform. Secondly, this transform produces an IBKE with KGC-FS. This is because the KGC keys are strictly used for authentication, and never for key derivation. Thirdly, as stated above the transform is slightly stricter than necessary regarding the ordering of events. In particular, certificates can be sent at any point in the protocol. If AKE is responder-concealing, then the transformed protocol may have the initiator send its certificate at the very beginning, and the responder send their certificate at the very end of the protocol, encrypted with the session key, thus preserving the responder-concealing property.[7]

[7]It is possible to go even further with this idea. Some protocols which are responder-

3.3.3 Security argument

We now informally discuss the security of LATKE before stating the main security theorem. To build intuition, we explain why each security assumption is necessary for LATKE.

SK-security. The base security of the IBKE prevents the most basic attacks, e.g., determining the session key of a passively observed session, impersonating a party who has not been compromised, or forcing two unrelated sessions to have the same session key.

Key-compromise impersonation resistance. If an adversary, Mallory, compromises Alice, then the only party Mallory should be able to impersonate (barring a brute-force attack) is Alice herself. In particular, she should not be able to impersonate Bob to Alice. This is captured by our KCIR requirement.

Full forward secrecy. Mallory can use a stolen mpk to corrupt the initial PAKE and passively observe the *cleartext* IBKE execution between two uncompromised parties, Alice and Bob. If Mallory records this execution, cracks the password associated with mpk (thus getting msk), and later corrupts Alice and Bob, she should still not be able to later determine the session key. This is captured by our full FS requirement.

OnlineComparePwd. The above scenario also elucidates the reason $\mathsf{OnlineComparePwd}$ is necessary in our ideal functionality. If the UC simulator is to simulate the cleartext IBKE session, it must know whether the stolen mpk matches one of the parties' PAKE inputs. This is ordinarily captured by the $\mathsf{Impersonate}$ procedure, which forces the target party to believe the stolen mpk's owner is their peer. However, it is not immediately clear when Mallory performs the attack if she intends to impersonate a party or to merely pass messages transparently. Since the latter case would not result in a change of perceived peer, the UC simulator cannot call $\mathsf{Impersonate}$ at that moment, and must use $\mathsf{OnlineComparePwd}$ instead.

SIM*-AC-CCA security and programmable ROM. As with $\mathsf{OnlineComparePwd}$, we require this symmetric security notion and ideal primitive for UC simulatability reasons. Suppose Alice and Bob are both uncompromised and executing the protocol. In this case, the simulator does not know anything about the parties' mpk: they may be unequal, equal to each other, equal to other parties with known passwords/mpk, or none of the above. Suppose Mallory waits until the first party, wlog Bob, outputs a session key K'' and sends the final (encrypted) message c to Alice. Then Mallory may immediately corrupt Alice, forcing the simulator to create an internal state which can explain c in the context of the protocol, and also demonstrate that the session key is indeed K''. The difficulty here is twofold: first, the simulator must find a message m that

concealing (resp. initiator-concealing) are also passively initiator-concealing (resp. passively responder concealing). An example is SIGMA-R [Kra03]. The transform described here does not preserve the passive concealing property. But if the protocol is modified so certificates are sent at the exact same time and encrypted under the same key as the user-identifying information in the underlying protocol, the both these properties are preserved. We will do exactly this in our experiments (Section 4).

makes sense and results in the output K'', and second, the simulator must be able to provide a decryption key that opens c to m. The first issue is resolved by the random oracle at the end of the protocol: the simulator can construct any transcript it wants, and simply program $\mathsf{H}_1(K', \mathsf{tr}) = K''$.[8] The second issue is resolved by our choice of a non-committing encryption primitive, which achieves SIM*-AC-CCA security.

Main theorem. We use three random oracles: H_0 is used for hashing passwords, H_1 is used for the final session key computation, and H_2 is used as for the symmetric ratchet in EUE. Further, we require whichever ideal primitive P that is required by the underlying SIM*-AC-CCA-secure authenticated encryption scheme (for AES-CTR + HMAC, for example, this is an ideal cipher and a random oracle). The proof can be found in Appendix B.

Theorem 1. *Let* IBKE *be a post-specified peer IBKE with SK-security with KCIR and full FS in the identity-based CK, eCK,* CK_{HMQV}*, or* CK^{+} *model. Let* AE^{P} *be a SIM*-AC-CCA-secure symmetric encryption scheme with ideal primitive* P*. Then the LATKE*^{post} *protocol realizes* $\mathcal{F}_{\mathsf{iPAKE}}^{\mathrm{ocw}}$ *in the* $(\mathcal{F}_{\mathsf{PAKE}}, \mathcal{F}_{\mathsf{RO}}, \mathsf{P})$*-hybrid model with erasure and adaptive corruptions.*

Similarly, if IBKE *is a pre-specified peer IBKE with SK-security in these models, then LATKE*^{pre} *realizes* $\mathcal{F}_{\mathsf{iPAKE}}^{\mathrm{ocw}}$ *in the* $(\mathcal{F}_{\mathsf{PAKE}}, \mathcal{F}_{\mathsf{RO}}, \mathsf{P})$*-hybrid model with erasure and adaptive corruptions.*

We note that the theorem accepts a range of IBKE models whose security we know to be incomparable **??**. In our proof, we avoid issues by using the narrowest set of adversary powers possible, a set which all models happen to share. In fact, the proof does not even require an IBKE security model with a notion of ephemeral key revelation. This permits a wider class of IBKE constructions, e.g., those constructed from AKEs with the forward secrecy transform from [BN11]. Our reduction is helped by the coarseness of the UC iPAKE model, which has strong session identifiers and only two types of corruption—long-term key corruption and total corruption.

Finally, our proof is written for LATKE$^{\mathrm{post}}$, but it is not dependent on the pre-specified peer model, and transfers with little modification (given in Appendix B). Thus, we achieve a general result for any peer model and any commonly used notion of IBKE security.

[8] This is resolved differently by Canetti and Krawczyk in their paper tying UC authenticated key exchange to SK-security [CK02b]. They define the *ACK property*—an AKE has the ACK property iff, once the first party outputs the session key, the internal states of both parties are simulatable using only the session key and public information. They show that this property is necessary to achieve UC simulatability, and describe a simple, generic transform to endow an SK-secure AKE with the ACK property, at the cost of one communication round. We are able to circumvent this overhead by using a programmable random oracle on our session key outputs. Since we must use a programmable model for iPAKE regardless [Hes20], this is essentially free.

4 Experiments

To demonstrate the practicality of LATKE, we instantiate it using a variety of PAKEs and IBKEs with different cryptographic assumptions and round complexities, and perform microbenchmarks on the resulting protocols. In particular, we instantiate the first post-quantum, identity-concealing iPAKE. For fair comparison, we also instantiate CHIP with the same underlying primitives where applicable. All our primitives are chosen to meet 128-bit security.

4.1 Choosing Primitives

In this section we list the PAKEs, IBKEs, and other primitives we chose for experimentation. For hashing, message authentication, and key derivation we use the Blake2b [ANWW13], HMAC [BCK96], and HKDF [Kra10] functions, respectively. For SIM*-AC-CCA-secure encryption, we use the encrypt-then-MAC construction described in Section 2.5 with the ChaCha20 stream cipher [Ber08] and the HMAC-Blake2b MAC. For simple Diffie-Hellman key agreement, we use the X25519 key-exchange protocol [Ber06]. For any protocol requiring prime-order group operations, we use the Ristretto255 group [VGH$^+$23].

4.1.1 PAKEs

We now describe the PAKEs we used to instantiate LATKE and CHIP.

`Cpace`**.** For our pre-quantum PAKE, we use CPace [HL19, AHH21], a one-round Diffie-Hellman-based construction. We also implemented and benchmarked KC-SPAKE2 [Sho20], but found it had higher communication cost, higher round complexity, and worse runtime efficiency. Thus, we omit KC-SPAKE2 entirely from our analysis.

`Cake`**.** For our post-quantum PAKE, we use CAKE [BCP$^+$23], a three-round generic construction which can be instantiated from any fuzzy, key-anonymous IND-CPA-secure KEM and any keyed permutation (standing in for an ideal cipher). For the keyed permutation, we use Kravatte, an instantiation of the Farfalle wide block cipher over the Keccak permutation [BDH$^+$17]. For the KEM, we use Saber (*Lightsaber* when using the 128-bit security parameters) [DKRV18],[9] whose ciphertexts and public keys pack perfectly into bytes, and so is compatible with CAKE with a wide-block cipher-based permutation.

4.1.2 IBKE

We now describe the IBKEs we used to instantiate LATKE and CHIP. We remark that we benefit greatly from the generality of Theorem 1. The protocols used below are proven in the id-CK, CK, eCK, and $\mathrm{CK}_{\mathrm{HMQV}}$ models, respectively.

[9] Saber is very similar to Kyber [BDK$^+$18] in construction, with the only essential differences being use of a power-of-two modulus rather than a prime modulus, and reliance on the Module Learning with Rounding (MLWR) problem rather than Module Learning with Errors (MLWE). Their similarity allows CAKE's required fuzziness and key-anonymity properties proven about Kyber in [BCP$^+$23, Lemma 2] to transfer to Saber.

`Fg(C)`. For more direct comparison with CHIP, we use the same Fiore-Gennaro IBKE that CHIP uses [FG10]. However, the IBKE only provides weak forward secrecy. To make it usable for LATKE, we add key confirmation to both sides using a MAC, as described in Section 3.3.2. We call the key confirmation variant `FgC`.

`IdSigmaR(Ed25519/Dilithium2)`. For identity-concealing and post-quantum security, we instantiate LATKE with the SIGMA-R responder-concealing protocol, due to Krawczyk [Kra03]. This is the only post-specified peer AKE we test. We apply the modification described by Peikert [Pei14] to make the protocol use a generic IND-CPA-secure KEM, rather than Diffie-Hellman. Finally, we apply our AKE-to-IBKE transform to the protocol, while preserving responder concealment. We illustrate the protocol in full in Figure 6. SIGMA-R and the IBKE transform both require signatures, so we benchmark with respect to the Ed25519 [BDL$^+$12] and Dilithium [BDK$^+$] signature schemes.[10]

`IdSigDh`. We also instantiate LATKE with the one-round AKE due to Bergsma et al. [BJS15], which is, at its core, signed Diffie-Hellman. We apply the same AKE-to-IBKE transform, using Ed25519 signatures, to this protocol.

`HmqvC`. Finally, we instantiate LATKE with the HMQV-C protocol (i.e., HMQV with key confirmation), described by Krawczyk [Kra05]. We apply the same transformation, also using Ed25519 signatures.

4.2 Experimental setup

We now describe the hardware, software, and methodology of our benchmarks.

Hardware. Since the intended use case for iPAKE is in mesh networking protocols, we chose to conduct benchmarks on a commodity WiFi router. The router, a Linksys E8450 AX3200, has a 64-bit ARM Cortex-A53 CPU (late 2012), and runs OpenWrt [ope] snapshot r17758-b118efa0d2 (late 2021).

Software. All benchmarks were written in Rust, in a total of 2.9kloc.[11] We used the Criterion benchmarking framework to measure performance, and disabled SHA2 hardware acceleration. SIMD is not supported on the target chipset, and we did not use any other form of parallel execution.

Methodology. For each combination of CHIP / LATKE with PAKE and IBKE, we measured *online runtime*, i.e., how long it takes to run the online portion of the protocol from beginning to end, ignoring any communication costs. We separately measured communication costs. To have a fair comparison of the included pre- and post-specified peer protocols, we exclude the communication costs that come from sending identifier strings (this is at most 64B per protocol execution). We also measure *setup runtime*, i.e., the time it takes to run $\mathsf{StorePwdFile}$, which is a one-time offline cost per device. To facilitate comparison between schemes, we

[10] There is good reason to consider using a pre-quantum signature scheme in an otherwise post-quantum protocol. The *store now, decrypt later* threat model only applies to encryption that may be broken in the future, not authentication. Thus, it suffices to use pre-quantum signatures until a cryptographically relevant quantum computer is imminent.

[11] Code is available at `https://github.com/rozbb/latke-ipake`

```
StorePwdFile(pw, id)
1 :  $(\mathsf{mpk}, \mathsf{msk}) := \mathsf{SigKg}(1^\lambda; \mathsf{H}_0(\mathsf{pw}))$
2 :  $(\mathsf{upk}, \mathsf{usk}) \leftarrow \mathsf{SigKg}(1^\lambda)$
3 :  $\sigma \leftarrow \mathsf{Sign}_{\mathsf{msk}}(\mathsf{upk} \| \mathsf{id})$
4 :  $\mathsf{cert} := (\mathsf{upk}, \mathsf{id}, \sigma)$
5 :  $\mathsf{pwfile} := (\mathsf{mpk}, \mathsf{cert}, \mathsf{usk})$
6 :  return pwfile
```

Alice$(\mathsf{pwfile}_A, \mathsf{sid}, \mathsf{ssid})$		**Bob**$(\mathsf{pwfile}_B, \mathsf{sid}, \mathsf{ssid})$
	.... Begin CAKE over mpk	
$(\mathsf{epk}, \mathsf{esk}) \leftarrow \mathsf{KemKg}(1^\lambda)$		
$\widetilde{\mathsf{epk}} \leftarrow \mathsf{E}_{\mathsf{mpk}\|\mathsf{sid}\|\mathsf{ssid}}(\mathsf{epk})$	$\xrightarrow{\widetilde{\mathsf{epk}}}$	
		$\mathsf{epk} := \mathsf{D}_{\mathsf{mpk}\|\mathsf{sid}\|\mathsf{ssid}}(\widetilde{\mathsf{epk}})$ $(\mathsf{ek}, K) \leftarrow \mathsf{Encap}(\mathsf{epk})$
	$\xleftarrow{\widetilde{\mathsf{ek}}}$	$\widetilde{\mathsf{ek}} := \mathsf{E}_{\mathsf{mpk}\|\mathsf{sid}\|\mathsf{ssid}}(\mathsf{ek})$
$\mathsf{ek} := \mathsf{D}_{\mathsf{mpk}\|\mathsf{sid}\|\mathsf{ssid}}(\widetilde{\mathsf{ek}})$ $K := \mathsf{Decap}_{\mathsf{esk}}(\mathsf{ek})$		
	.... Begin SIGMA-R, encrypted with K	
$\mathsf{nonce}_A \leftarrow \{0,1\}^\lambda$	EUE_K	$\mathsf{nonce}_B \leftarrow \{0,1\}^\lambda$
$(\mathsf{epk}, \mathsf{esk}) \leftarrow \mathsf{KemKg}(1^\lambda)$	$\xrightarrow{\mathsf{epk}\ \mathsf{nonce}_A}$	
	$\xleftarrow{\mathsf{ek}\ \mathsf{nonce}_B}$	$(\mathsf{ek}, \mathsf{ss}) \leftarrow \mathsf{Encap}(\mathsf{epk})$
$\mathsf{ss} := \mathsf{Decap}_{\mathsf{esk}}(\mathsf{ek})$		
$(K', K^{(e)}, K^{(m)}) := \mathsf{KDF}(\mathsf{ss})$		$(K', K^{(e)}, K^{(m)}) := \mathsf{KDF}(\mathsf{ss})$
$\sigma_A \leftarrow \mathsf{Sign}_{\mathsf{usk}_A}(0\|\mathsf{nonce}_B\|\mathsf{sid}\|\mathsf{ssid}$ $\|\mathsf{epk}\|\mathsf{ek}\|\mathsf{cert}_A)$		$\sigma_B \leftarrow \mathsf{Sign}_{\mathsf{usk}_B}(1\|\mathsf{nonce}_A\|\mathsf{sid}\|\mathsf{ssid}$ $\|\mathsf{ek}\|\mathsf{epk}\|\mathsf{cert}_B)$
$\tau_A := \mathsf{Mac}_{K^{(m)}}(0\|\mathsf{sid}\|\mathsf{ssid}\|\mathsf{id}_A)$		$\tau_B := \mathsf{Mac}_{K^{(m)}}(1\|\mathsf{sid}\|\mathsf{ssid}\|\mathsf{id}_A)$
$c_A := \mathsf{Enc}_{K^{(e)}}(\mathsf{cert}_A\|\sigma_A\|\tau_A)$	$\xrightarrow{c_A}$	$c_B := \mathsf{Enc}_{K^{(e)}}(\mathsf{cert}_B\|\sigma_B\|\tau_B)$
		decrypt c_A or **abort**
	$\xleftarrow{c_B}$	**verify** $\mathsf{cert}_A, \sigma_A, \tau_A$ wrt $\mathsf{mpk}, \mathsf{upk}_A, K^{(m)}$
decrypt c_B or **abort**		
verify $\mathsf{cert}_B, \sigma_B, \tau_B$ wrt $\mathsf{mpk}, \mathsf{upk}_B, K^{(m)}$		
$K'' := \mathsf{H}_1(K', \mathsf{tr})$		$K'' := \mathsf{H}_1(K', \mathsf{tr})$
return (id_B, K'') or $\perp$ on EUE err		**return** (id_A, K'') or $\perp$ on EUE err

Figure 6: LATKE$^{\mathrm{post}}$ instantiated with the CAKE PAKE [BCP$^+$23], and the SIGMA-R AKE [Kra03, Pei14] with identity certificates (Section 3.3.2). E, D represents a wide-block cipher.

iPAKE	PQ?	Hiding?	Rounds	Comm.	Setup	Online	Online/CHIP
`Chip[Cpace,Fg]` [CNPR22]	×	×	2	208B	284µs	5.33ms	1×
$\texttt{Latke}^{\texttt{pre}}$`[Cpace,FgC]`	×	×	4	404B	314µs	5.56ms	1.04×
$\texttt{Latke}^{\texttt{pre}}$`[Cpace,IdHmqvC]`	×	×	4	532B	467µs	5.62ms	1.05×
$\texttt{Latke}^{\texttt{pre}}$`[Cpace,IdSigDh]`	×	×	2	616B	615µs	8.32ms	1.56×
$\texttt{Latke}^{\texttt{post}}$`[Cake,IdSigmaREd25519]`	enc.	✓	6	3.53kB	813µs	5.46ms	1.03×
$\texttt{Latke}^{\texttt{post}}$`[Cake,IdSigmaRDilithium2]`	enc.+auth.	✓	6	15.5kB	2.55ms	10.1ms	1.89×

Table 1: Performance characteristics of CHIP and LATKE, instantiated with varying IBKEs and PAKEs. Reported online latencies are for the full protocol, excluding communication times.

do not use any hard password hashing functions in StorePwdFile (e.g., Argon2 or PBKDF2), though any realistic implementation must use one.

We report the medians of the recorded latencies. Across all benchmarks, then maximum observed relative standard error of the median was 0.5%.

4.3 Discussion

We show the results of our benchmarks in Table 1.

As expected, LATKE$^{\text{pre}}$ with `FgC` performs closely to CHIP with `Fg`, both in setup and online time. The additional communication overhead of 196B can be attributed to the additional key confirmation information that CHIP does not require, plus the ciphertext expansion due to EUE.

The `HmqvC` instantiation of LATKE performs similarly well, albeit with a slightly higher setup time and communication cost. The round-optimal instantiation with `IdSigDh` performs noticeably worse due to its reliance on signatures for authentication, rather than the implicit authentication of `FgC` and `HmqvC`.

The `IdSigmaREd25519` post-quantum encryption is surprisingly close to CHIP in online runtime, albeit with nearly 17× the communication cost. And, as expected, `IdSigmaRDilithium2` post-quantum encryption and authentication instantiation performed the worst across the board, with over 74× the communication cost compared to CHIP, but surprisingly only 1.89× the online runtime.

As a possible extension, we also measured an optimization whereby every user saves the certificate of the other party. In future exchanges, the user does not have to perform a signature verification. This saves 822µs on average, making $\texttt{Latke}^{\texttt{pre}}$`[Cpace,IdHmqvC]` and $\texttt{Latke}^{\texttt{post}}$`[Cake,IdSigmaREd25519]` faster than `Chip[Cpace,Fg]` on every execution after the first (again, ignoring communication costs).

Finally, we note there are still performance gains to be had for the post-quantum SIGMA-R instantiation. The reason we use Saber for IBKE is simply for convenience in testing, since it's already used in CAKE. But there are post-quantum KEMs faster than Saber, such NTTRU [LS19] and Kyber [BDK+18].

5 Future work

Post-quantum siPAKE. Recall the CRISP protocol is a *strong* iPAKE (siPAKE), meaning that it is robust to precomputation attacks. In CHIP and LATKE, an adversary who precomputes a table of mpk values from various password guesses can use it to speed up a brute-force attack, should they ever retrieve an mpk from a user device. The CRISP construction uses algebraic techniques which rely on bilinear pairings in order to locally rerandomize its pwfile without breaking authenticity. It is not immediately clear if these properties can be obtained generically with an AKE and PAKE, nor if there exists a post-quantum analogue to the local rerandomization and pairing steps.

In addition, we reiterate the call for future work by the authors of CHIP/CRISP, that it would be useful to find a local rerandomization step that produces a problem for an attacker with tweakable hardness. Currently, CRISP forces the attacker to compute 1 pairing per guess—on the order of 1ms—whereas an OPRF-based strong construction like OPAQUE force an attacker to compute a hard hash per guess—on the order of 100ms.

Hybrid iPAKE. We have described LATKE using entirely post-quantum and entirely pre-quantum primitives, but the common path taken recently by standardization bodies is to provide *hybrid* schemes, which combine two schemes and enjoy the security of the harder of the two, even if we don't yet know which one that is. Since there already exist hybrid IBKEs with the properties necessary for LATKE, all that remains to build a hybrid iPAKE is to find a hybrid PAKE.

To the authors' knowledge, there is no known hybrid PAKE, let alone a generic method for creating one. Consider the concatenation transform, used in KEMs, whereby two KEMs are hybridized by simply running them in parallel, concatenating the final shared secrets, and hashing the concatenation. If the PAKEs rely on a computational indistinguishability assumption for passive security, as CAKE does (specifically, LWE for fuzziness and key anonymity), then the hybrid PAKE's passive security is only as secure as the *least* secure of the two underlying PAKEs.

Round-optimal post-quantum LATKE. As demonstrated in Section 4, pre-quantum LATKE can be achieved using only 2 rounds of communication, though this was only shown using a pre-quantum PAKE and IBKE. It remains an open question whether the lower bound can be achieved using post-quantum building blocks.

One-round PAKEs from isogeny assumptions are known [AEK$^+$22, IY23], but they are only proved secure in (an extension of) the game-based model of Bellare, Pointcheval, and Rogaway [BPR00]. It is not immediately clear if this is sufficient for usage in LATKE, or if they can be proven to UC-realize $\mathcal{F}_{\mathsf{PAKE}}$.

One-round post-quantum IBKEs are also difficult to instantiate. We can consider just AKE in our round complexity analysis, since our AKE-to-IBKE transform does not add rounds. Note a one-round AKE is equivalently a non-interactive key exchange (NIKE), since we can simply make the first round an exchange of the parties' public keys.

The only post-quantum AKE considered in Section 4 is SIGMA-R [Kra03], which has four rounds of communication. Schemes with two rounds are well known [BCNS15, ADPS16], but there are few schemes with just one round of communication. The CSIDH NIKE [CLM$^+$18], based on an isogeny graph walk assumption, appears to be the only plausibly efficient protocol in this space. The only proposed lattice-based NIKE, Swoosh [GdKQ$^+$23], requires large lattice dimensions (hence large communication costs) and its actively secure form requires each party to compute a non-interactive zero-knowledge proof (which also must be post-quantum) for every key exchange.

Finally, we again reiterate the call for future work by the CHIP/CRISP authors, that it would be useful to find a one-round iPAKE, or else prove that one cannot be constructed.

6 Acknowledgements

Many thanks to Mark Schultz for inspiring conversations. Thanks to Levi Schuck, Greg Rubin, Sanketh Menda, Doruk Gür, Cas Cremers, Shahar Paz, Julia Hesse, and Steve Thomas for providing feedback and answering questions.

References

[ADPS16] Erdem Alkim, Léo Ducas, Thomas Pöppelmann, and Peter Schwabe. Post-quantum key exchange - A new hope. In Thorsten Holz and Stefan Savage, editors, *USENIX Security 2016*, pages 327–343. USENIX Association, August 2016.

[AEK$^+$22] Michel Abdalla, Thorsten Eisenhofer, Eike Kiltz, Sabrina Kunzweiler, and Doreen Riepel. Password-authenticated key exchange from group actions. In Dodis and Shrimpton [DS22], pages 699–728. `doi:10.1007/978-3-031-15979-4_24`.

[AHH21] Michel Abdalla, Björn Haase, and Julia Hesse. Security analysis of CPace. In Mehdi Tibouchi and Huaxiong Wang, editors, *ASIACRYPT 2021, Part IV*, volume 13093 of *LNCS*, pages 711–741. Springer, Heidelberg, December 2021. `doi:10.1007/978-3-030-92068-5_24`.

[All22a] Connectivity Standards Alliance. Matter Specification v1.0, September 2022. URL: `https://csa-iot.org/wp-content/uploads/2022/11/22-27349-001_Matter-1.0-Core-Specification.pdf`.

[All22b] The Wifi Alliance. WPA3 Specification v3.1. 2022. URL: `https://www.wi-fi.org/system/files/WPA3%20Specification%20v3.1.pdf`.

[ANWW13] Jean-Philippe Aumasson, Samuel Neves, Zooko Wilcox-O'Hearn, and Christian Winnerlein. BLAKE2: Simpler, smaller, fast as MD5. In Michael J. Jacobson Jr., Michael E. Locasto, Payman Mohassel, and Reihaneh Safavi-Naini, editors, *ACNS 13*, volume 7954 of *LNCS*, pages 119–135. Springer, Heidelberg, June 2013. `doi:10.1007/978-3-642-38980-1_8`.

[App] Apple. Apple Platform Security, May 2022. URL: `https://help.apple.com/pdf/security/en_US/apple-platform-security-guide.pdf`.

[Bas22] Andrea Basso. Poster: A post-quantum oblivious prf from isogenies. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, CCS '22, page 3327–3329, New York, NY, USA, 2022. Association for Computing Machinery. `doi:10.1145/3548606.3563542`.

[Bas23] Andrea Basso. A post-quantum round-optimal oblivious PRF from isogenies. Cryptology ePrint Archive, Report 2023/225, 2023. `https://eprint.iacr.org/2023/225`.

[BCK96] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. Keying hash functions for message authentication. In Neal Koblitz, editor, *CRYPTO'96*, volume 1109 of *LNCS*, pages 1–15. Springer, Heidelberg, August 1996. `doi:10.1007/3-540-68697-5_1`.

[BCNS15] Joppe W. Bos, Craig Costello, Michael Naehrig, and Douglas Stebila. Post-quantum key exchange for the TLS protocol from the ring learning with errors problem. In *2015 IEEE Symposium on Security and Privacy*, pages 553–570. IEEE Computer Society Press, May 2015. `doi:10.1109/SP.2015.40`.

[BCP$^+$23] Hugo Beguinet, Céline Chevalier, David Pointcheval, Thomas Ricosset, and Mélissa Rossi. Get a cake: Generic transformations from key encaspulation mechanisms to password authenticated key exchanges. In Mehdi Tibouchi and XiaoFeng Wang, editors, *Applied Cryptography and Network Security*, pages 516–538, Cham, 2023. Springer Nature Switzerland.

[BDF$^+$11] Dan Boneh, Özgür Dagdelen, Marc Fischlin, Anja Lehmann, Christian Schaffner, and Mark Zhandry. Random oracles in a quantum world. In Dong Hoon Lee and Xiaoyun Wang, editors, *ASIACRYPT 2011*, volume 7073 of *LNCS*, pages 41–69. Springer, Heidelberg, December 2011. `doi:10.1007/978-3-642-25385-0_3`.

[BDH$^+$17] Guido Bertoni, Joan Daemen, Seth Hoffert, Michaël Peeters, Gilles Van Assche, and Ronny Van Keer. Farfalle: parallel permutation-based cryptography. *IACR Trans. Symm. Cryptol.*, 2017(4):1–38, 2017. `doi:10.13154/tosc.v2017.i4.1-38`.

[BDK$^+$] Shi Bai, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, Peter Schwabe, Gregor Seiler, and Damien Stehlé. CRYSTALS-Dilithium. URL: `https://pq-crystals.org/dilithium/data/dilithium-specification-round3-20210208.pdf`.

[BDK15] Alex Biryukov, Daniel Dinu, and Dmitry Khovratovich. Argon2: the memory-hard function for password hashing and other applications. 2015.

[BDK$^+$18] Joppe Bos, Leo Ducas, Eike Kiltz, T Lepoint, Vadim Lyubashevsky, John M. Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehle. Crystals - kyber: A cca-secure module-lattice-based kem. In *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 353–367, 2018. `doi:10.1109/EuroSP.2018.00032`.

[BDL$^+$12] Daniel J. Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. High-speed high-security signatures. *Journal of Cryptographic Engineering*, 2(2):77–89, September 2012. `doi:10.1007/s13389-012-0027-1`.

[Ber06] Daniel J. Bernstein. Curve25519: New Diffie-Hellman speed records. In Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin, editors, *PKC 2006*, volume 3958 of *LNCS*, pages 207–228. Springer, Heidelberg, April 2006. `doi:10.1007/11745853_14`.

[Ber08] Daniel J. Bernstein. The ChaCha family of stream ciphers, January 2008. URL: `https://cr.yp.to/chacha.html`.

[BJS15] Florian Bergsma, Tibor Jager, and Jörg Schwenk. One-round key exchange with strong security: An efficient and generic construction in the standard model. In Jonathan Katz, editor, *PKC 2015*, volume 9020 of *LNCS*, pages 477–494. Springer, Heidelberg, March / April 2015. `doi:10.1007/978-3-662-46447-2_21`.

[BKM$^+$21] Andrea Basso, Péter Kutas, Simon-Philipp Merz, Christophe Petit, and Antonio Sanso. Cryptanalysis of an oblivious PRF from supersingular isogenies. In Mehdi Tibouchi and Huaxiong Wang, editors, *ASIACRYPT 2021, Part I*, volume 13090 of *LNCS*, pages 160–184. Springer, Heidelberg, December 2021. `doi:10.1007/978-3-030-92062-3_6`.

[BKW20] Dan Boneh, Dmitry Kogan, and Katharine Woo. Oblivious pseudorandom functions from isogenies. In Shiho Moriai and Huaxiong Wang, editors, *ASIACRYPT 2020, Part II*, volume 12492 of *LNCS*, pages 520–550. Springer, Heidelberg, December 2020. `doi:10.1007/978-3-030-64834-3_18`.

[BLR04] Boaz Barak, Yehuda Lindell, and Tal Rabin. Protocol initialization for the framework of universal composability. Cryptology ePrint Archive, Report 2004/006, 2004. https://eprint.iacr.org/2004/006.

[BM92] Steven M. Bellovin and Michael Merritt. Encrypted key exchange: Password-based protocols secure against dictionary attacks. In *1992 IEEE Symposium on Security and Privacy*, pages 72–84. IEEE Computer Society Press, May 1992. doi:10.1109/RISP.1992.213269.

[BN11] Colin Boyd and Juan González Nieto. On forward secrecy in one-round key exchange. In Liqun Chen, editor, *Cryptography and Coding*, pages 451–468, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[Bon03] Dan Boneh, editor. *CRYPTO 2003*, volume 2729 of *LNCS*. Springer, Heidelberg, August 2003.

[BPR00] Mihir Bellare, David Pointcheval, and Phillip Rogaway. Authenticated key exchange secure against dictionary attacks. In Bart Preneel, editor, *EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 139–155. Springer, Heidelberg, May 2000. doi:10.1007/3-540-45539-6_11.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, October 2001. doi:10.1109/SFCS.2001.959888.

[CC07a] Zhaohui Cheng and Liqun Chen. On security proof of mccullaghbarreto's key agreement protocol and its variants. *International Journal of Security and Networks*, 2(3-4):251–259, 2007. doi:10.1504/IJSN.2007.013178.

[CC07b] Sherman S. M. Chow and Kim-Kwang Raymond Choo. Strongly-secure identity-based key agreement and anonymous extension. In Juan A. Garay, Arjen K. Lenstra, Masahiro Mambo, and René Peralta, editors, *Information Security*, pages 203–220, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[CJSV22] Ran Canetti, Palak Jain, Marika Swanberg, and Mayank Varia. Universally composable end-to-end secure messaging. In Dodis and Shrimpton [DS22], pages 3–33. doi:10.1007/978-3-031-15979-4_1.

[CK01] Ran Canetti and Hugo Krawczyk. Analysis of key-exchange protocols and their use for building secure channels. In Birgit Pfitzmann, editor, *EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 453–474. Springer, Heidelberg, May 2001. doi:10.1007/3-540-44987-6_28.

[CK02a] Ran Canetti and Hugo Krawczyk. Security analysis of IKE's signature-based key-exchange protocol. In Moti Yung, editor, *CRYPTO 2002*, volume 2442 of *LNCS*, pages 143–161. Springer, Heidelberg, August 2002. https://eprint.iacr.org/2002/120/. doi:10.1007/3-540-45708-9_10.

[CK02b] Ran Canetti and Hugo Krawczyk. Universally composable notions of key exchange and secure channels. In Lars R. Knudsen, editor, *EUROCRYPT 2002*, volume 2332 of *LNCS*, pages 337–351. Springer, Heidelberg, April / May 2002. doi:10.1007/3-540-46035-7_22.

[CLM$^+$18] Wouter Castryck, Tanja Lange, Chloe Martindale, Lorenz Panny, and Joost Renes. CSIDH: An efficient post-quantum commutative group action. In Thomas Peyrin and Steven Galbraith, editors, *ASIACRYPT 2018, Part III*, volume 11274 of *LNCS*, pages 395–427. Springer, Heidelberg, December 2018. doi:10.1007/978-3-030-03332-3_15.

[CNPR22] Cas Cremers, Moni Naor, Shahar Paz, and Eyal Ronen. CHIP and CRISP: Protecting all parties against compromise through identity-binding PAKEs. In Dodis and Shrimpton [DS22], pages 668–698. doi:10.1007/978-3-031-15979-4_23.

[Cre11] Cas Cremers. Examining indistinguishability-based security models for key exchange protocols: the case of CK, CK-HMQV, and eCK. In Bruce S. N. Cheung, Lucas Chi Kwong Hui, Ravi S. Sandhu, and Duncan S. Wong, editors, *ASIACCS 11*, pages 80–91. ACM Press, March 2011.

[DFG$^+$23] Gareth T. Davies, Sebastian H. Faller, Kai Gellert, Tobias Handirk, Julia Hesse, Máté Horváth, and Tibor Jager. Security analysis of the WhatsApp end-to-end encrypted backup protocol. In Handschuh and Lysyanskaya [HL23], pages 330–361. doi:10.1007/978-3-031-38551-3_11.

[DKRV18] Jan-Pieter D'Anvers, Angshuman Karmakar, Sujoy Sinha Roy, and Frederik Vercauteren. Saber: Module-LWR based key exchange, CPA-secure encryption and CCA-secure KEM. In Antoine Joux, Abderrahmane Nitaj, and Tajjeeddine Rachidi, editors, *AFRICACRYPT 18*, volume 10831 of *LNCS*, pages 282–305. Springer, Heidelberg, May 2018. doi:10.1007/978-3-319-89339-6_16.

[Dod23] Lucas Dodgson. Post-Quantum Building Blocks for Secure Computation – the Legendre OPRF, September 2023. URL: https://ethz.ch/content/dam/ethz/special-interest/infk/inst-infsec/appliedcrypto/education/theses/Master_Thesis_Post_Quantum_Building_blocks_for_secure_computation.pdf.

[DS22] Yevgeniy Dodis and Thomas Shrimpton, editors. *CRYPTO 2022, Part II*, volume 13508 of *LNCS*. Springer, Heidelberg, August 2022.

[FG10] Dario Fiore and Rosario Gennaro. *Identity-Based Key Exchange Protocols without Pairings*, pages 42–77. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010. doi:10.1007/978-3-642-17499-5_3.

[Fil18] Rick Fillion. Secure Remote Password (SRP): How 1Password uses it, 2018. Section: 1Password. URL: https://blog.1password.com/developers-how-we-use-srp-and-you-can-too/.

[FOO23] Sebastian Faller, Astrid Ottenhues, and Johannes Ottenhues. Composable oblivious pseudo-random functions via garbled circuits. In Abdelrahaman Aly and Mehdi Tibouchi, editors, *Progress in Cryptology – LATINCRYPT 2023*, pages 249–270, Cham, 2023. Springer Nature Switzerland.

[FSXY12] Atsushi Fujioka, Koutarou Suzuki, Keita Xagawa, and Kazuki Yoneyama. Strongly secure authenticated key exchange from factoring, codes, and lattices. In Marc Fischlin, Johannes Buchmann, and Mark Manulis, editors, *PKC 2012*, volume 7293 of *LNCS*, pages 467–484. Springer, Heidelberg, May 2012. doi:10.1007/978-3-642-30057-8_28.

[GdKQ$^+$23] Phillip Gajland, Bor de Kock, Miguel Quaresma, Giulio Malavolta, and Peter Schwabe. Swoosh: Practical lattice-based non-interactive key exchange. Cryptology ePrint Archive, Report 2023/271, 2023. https://eprint.iacr.org/2023/271.

[GGJJ23] Kai Gellert, Kristian Gjøsteen, Håkon Jacobsen, and Tibor Jager. On optimal tightness for key exchange with full forward secrecy via key confirmation. In Handschuh and Lysyanskaya [HL23], pages 297–329. doi:10.1007/978-3-031-38551-3_10.

[GMR06] Craig Gentry, Philip MacKenzie, and Zulfikar Ramzan. A method for making password-based key exchange resilient to server compromise. In Cynthia Dwork, editor, *CRYPTO 2006*, volume 4117 of *LNCS*, pages 142–159. Springer, Heidelberg, August 2006. doi:10.1007/11818175_9.

[Gon93] Li Gong. Lower bounds on messages and rounds for network authentication protocols. In Dorothy E. Denning, Raymond Pyle, Ravi Ganesan, Ravi S. Sandhu, and Victoria Ashby, editors, *ACM CCS 93*, pages 26–37. ACM Press, November 1993. doi:10.1145/168588.168592.

[Gün90] Christoph G. Günther. An identity-based key-exchange protocol. In Jean-Jacques Quisquater and Joos Vandewalle, editors, *EUROCRYPT'89*, volume 434 of *LNCS*, pages 29–37. Springer, Heidelberg, April 1990. doi:10.1007/3-540-46885-4_5.

[Ham15] Mike Hamburg. [curves] SPAKE2 and SPAKE2 Elligator Edition, 2015. URL: `https://moderncrypto.org/mail-archive/curves/2015/000424.html`.

[Har08] Dan Harkins. Simultaneous authentication of equals: A secure, password-based key exchange for mesh networks. In *2008 Second International Conference on Sensor Technologies and Applications (sensorcomm 2008)*, pages 839–844, 2008. `doi:10.1109/SENSORCOMM.2008.131`.

[HC09] Hai Huang and Zhenfu Cao. An ID-based authenticated key exchange protocol based on bilinear Diffie-Hellman problem. In Wanqing Li, Willy Susilo, Udaya Kiran Tupakula, Reihaneh Safavi-Naini, and Vijay Varadharajan, editors, *ASIACCS 09*, pages 333–342. ACM Press, March 2009.

[Hes20] Julia Hesse. Separating symmetric and asymmetric password-authenticated key exchange. In Clemente Galdi and Vladimir Kolesnikov, editors, *SCN 20*, volume 12238 of *LNCS*, pages 579–599. Springer, Heidelberg, September 2020. `doi:10.1007/978-3-030-57990-6_29`.

[HL19] Björn Haase and Benoît Labrique. AuCPace: Efficient verifier-based PAKE protocol tailored for the IIoT. *IACR TCHES*, 2019(2):1–48, 2019. `https://tches.iacr.org/index.php/TCHES/article/view/7384`. `doi:10.13154/tches.v2019.i2.1-48`.

[HL23] Helena Handschuh and Anna Lysyanskaya, editors. *CRYPTO 2023, Part IV*, volume 14084 of *LNCS*. Springer, Heidelberg, August 2023.

[Hv22] Feng Hao and Paul C. van Oorschot. SoK: Password-authenticated key exchange - theory, practice, standardization and real-world lessons. In Yuji Suga, Kouichi Sakurai, Xuhua Ding, and Kazue Sako, editors, *ASIACCS 22*, pages 697–711. ACM Press, May / June 2022. `doi:10.1145/3488932.3523256`.

[IY23] Ren Ishibashi and Kazuki Yoneyama. Compact password authenticated key exchange from group actions. In Leonie Simpson and Mir Ali Rezazadeh Baee, editors, *ACISP 23*, volume 13915 of *LNCS*, pages 220–247. Springer, Heidelberg, July 2023. `doi:10.1007/978-3-031-35486-1_11`.

[Jab97] David P. Jablon. Extended password key exchange protocols immune to dictionary attacks. In *6th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE 1997)*, pages 248–255, Cambridge, MA, USA, June 18–20, 1997. IEEE Computer Society.

[Jae23] Joseph Jaeger. Let attackers program ideal models: Modularity and composability for adaptive compromise. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part III*, volume 14006 of *LNCS*, pages 101–131. Springer, Heidelberg, April 2023. `doi:10.1007/978-3-031-30620-4_4`.

[JKX18] Stanislaw Jarecki, Hugo Krawczyk, and Jiayu Xu. OPAQUE: An asymmetric PAKE protocol secure against pre-computation attacks. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part III*, volume 10822 of *LNCS*, pages 456–486. Springer, Heidelberg, April / May 2018. `doi:10.1007/978-3-319-78372-7_15`.

[JT20] Joseph Jaeger and Nirvan Tyagi. Handling adaptive compromise for practical encryption schemes. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part I*, volume 12170 of *LNCS*, pages 3–32. Springer, Heidelberg, August 2020. `doi:10.1007/978-3-030-56784-2_1`.

[KN09] Eike Kiltz and Gregory Neven. Identity-Based Signatures. 2009.

[Kra03] Hugo Krawczyk. SIGMA: The "SIGn-and-MAc" approach to authenticated Diffie-Hellman and its use in the IKE protocols. In Boneh [Bon03], pages 400–425. `doi:10.1007/978-3-540-45146-4_24`.

[Kra05] Hugo Krawczyk. HMQV: A high-performance secure Diffie-Hellman protocol. In Victor Shoup, editor, *CRYPTO 2005*, volume 3621 of *LNCS*, pages 546–566. Springer, Heidelberg, August 2005. `doi:10.1007/11535218_33`.

[Kra10] Hugo Krawczyk. Cryptographic extraction and key derivation: The HKDF scheme. In Tal Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 631–648. Springer, Heidelberg, August 2010. `doi:10.1007/978-3-642-14623-7_34`.

[LLHG23] Xiangyu Liu, Shengli Liu, Shuai Han, and Dawu Gu. EKE meets tight security in the Universally Composable framework. In Alexandra Boldyreva and Vladimir Kolesnikov, editors, *PKC 2023, Part I*, volume 13940 of *LNCS*, pages 685–713. Springer, Heidelberg, May 2023. `doi:10.1007/978-3-031-31368-4_24`.

[LLM07] Brian A. LaMacchia, Kristin Lauter, and Anton Mityagin. Stronger security of authenticated key exchange. In Willy Susilo, Joseph K. Liu, and Yi Mu, editors, *ProvSec 2007*, volume 4784 of *LNCS*, pages 1–16. Springer, Heidelberg, November 2007.

[Lou21] Karim Lounis. Cut it: Deauthentication attack on bluetooth. In *2021 14th International Conference on Security of Information* *and Networks (SIN)*, volume 1, pages 1–8, 2021. doi:10.1109/SIN54109.2021.9699265.

[LS19] Vadim Lyubashevsky and Gregor Seiler. NTTRU: Truly fast NTRU using NTT. *IACR TCHES*, 2019(3):180–201, 2019. https://tches.iacr.org/index.php/TCHES/article/view/8293. doi:10.13154/tches.v2019.i3.180-201.

[mag] Magic Wormhole. URL: https://github.com/magic-wormhole/magic-wormhole.

[Mar16] Moxie Marlinspike. The Double Ratchet Algorithm, 2016. URL: https://signal.org/docs/specifications/doubleratchet/.

[Oec03] Philippe Oechslin. Making a faster cryptanalytic time-memory trade-off. In Boneh [Bon03], pages 617–630. doi:10.1007/978-3-540-45146-4_36.

[Oka88] Eiji Okamoto. Key distribution systems based on identification information. In Carl Pomerance, editor, *CRYPTO'87*, volume 293 of *LNCS*, pages 194–202. Springer, Heidelberg, August 1988. doi:10.1007/3-540-48184-2_15.

[ope] Welcome to the OpenWrt Project. URL: https://openwrt.org.

[Pau22] Sebastian Paul. On the Transition to Post-Quantum Cryptography in the Industrial Internet of Things. May 2022.

[Pei14] Chris Peikert. Lattice cryptography for the internet. In Michele Mosca, editor, *Post-Quantum Cryptography - 6th International Workshop, PQCrypto 2014*, pages 197–219. Springer, Heidelberg, October 2014. doi:10.1007/978-3-319-11659-4_12.

[SE15] Jae Hong Seo and Keita Emura. Revocable hierarchical identity-based encryption: History-free update, security against insiders, and short ciphertexts. In Kaisa Nyberg, editor, *CT-RSA 2015*, volume 9048 of *LNCS*, pages 106–123. Springer, Heidelberg, April 2015. doi:10.1007/978-3-319-16715-2_6.

[Shi03] Kyungah Shim. Efficient id-based authenticated key agreement protocol based on weil pairing. *Electronics Letters*, 39:653–654(1), April 2003. URL: https://digital-library.theiet.org/content/journals/10.1049/el_20030448.

[Sho20] Victor Shoup. Security analysis of SPAKE2+. In Rafael Pass and Krzysztof Pietrzak, editors, *Theory of Cryptography*, pages 31–60, Cham, 2020. Springer International Publishing.

[SPT13] Craig A. Shue, Nathanael Paul, and Curtis R. Taylor. From an IP address to a street address: Using wireless signals to locate a target. In *7th USENIX Workshop on Offensive Technologies (WOOT 13)*, Washington, D.C., August 2013. USENIX Association. URL: `https://www.usenix.org/conference/woot13/workshop-program/presentation/shue`.

[SRV23] Domien Schepers, Aanjhan Ranganathan, and Mathy Vanhoef. Framing frames: Bypassing Wi-Fi encryption by manipulating transmit queues. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 53–68, Anaheim, CA, August 2023. USENIX Association. URL: `https://www.usenix.org/conference/usenixsecurity23/presentation/schepers`.

[Tho22a] Steve Thomas. bscrypt: A Cache Hard Password Hash, 2022. URL: `https://tobtu.com/files/bsideslv2022.pdf`.

[Tho22b] Steve Thomas. Demystifying Key Stretching and PAKEs. Black Hat 2022, Aug 2022. URL: `https://www.blackhat.com/us-22/briefings/schedule/#demystifying-key-stretching-and-pakes-27615`.

[Thr15] Thread Group. Thread commissioning, July 2015. URL: `https://www.threadgroup.org/Portals/0/documents/support/CommissioningWhitePaper_658_2.pdf`.

[VGH$^+$23] Henry de Valence, Jack Grigg, Mike Hamburg, Isis Lovecruft, George Tankersley, and Filippo Valsorda. The ristretto255 and decaf448 Groups. Request for Comments RFC 9496, Internet Engineering Task Force, December 2023. URL: `https://datatracker.ietf.org/doc/rfc9496`.

[Wan13] Yongge Wang. *Efficient Identity-Based and Authenticated Key Agreement Protocol*, pages 172–197. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013. `doi:10.1007/978-3-642-35840-1_9`.

[wif21] Ieee standard for information technology–telecommunications and information exchange between systems - local and metropolitan area networks–specific requirements - part 11: Wireless lan medium access control (mac) and physical layer (phy) specifications. *IEEE Std 802.11-2020 (Revision of IEEE Std 802.11-2016)*, pages 1–4379, 2021. `doi:10.1109/IEEESTD.2021.9363693`.

[You22] Shelanda D. Young. M-23-02 Memo on migrating to post-quantum cryptography, Nov. 18 2022. *Executive Office of the President, Office of Management and Budget.* URL: `https://www.whitehouse.gov/wp-content/uploads/2022/11/M-23-02-M-Memo-on-Migrating-to-Post-Quantum-Cryptography.pdf`.

A Deferred preliminaries

Here we include the preliminaries necessary for our main proof in Appendix B.

A.1 Notation

We say a function $f : \mathbb{N} \to \mathbb{R}$ is *negligible in* n, denoted $f(n) = \mathsf{negl}(n)$, iff $|f(n)| = n^{-\omega(1)}$ or, equivalently, for any $c > 0$, there is an n_0 such that $f(n) > n^{-c}$ for all $n > n_0$. We say a function $f : \mathbb{N} \to \mathbb{R}$ is *polynomial in* n, denoted $f(n) = \mathsf{poly}(n)$, iff for some $M, d > 0$, there is an n_0 such that $|f(n)| \leq Mn^c$ for all $n > n_0$.

A.2 Probability

A *probability ensemble* $\mathcal{S}$ is an infinite sequence of probability distributions $\mathcal{S}_1, \mathcal{S}_2, \ldots$, where each $\mathcal{S}_i$ is over a set $A_i \subset \{0,1\}^{\ell(i)}$, where $\ell(n) = \mathsf{poly}(n)$ is some length function. The *statistical distance* between two probability distributions $\mathcal{S}, \mathcal{S}'$ over sets A, A' is

$$\Delta(\mathcal{S}, \mathcal{S}') = \frac{1}{2} \sum_{x \in A \cup A'} |\Pr[y = x \mid y \leftarrow \mathcal{S}] - \Pr[y = x \mid y \leftarrow \mathcal{S}']|.$$

We say that two probability ensembles $\mathcal{S}, \mathcal{S}'$ are *statistically indistinguishable*, denoted $\mathcal{S} \overset{s}{\approx} \mathcal{S}'$ iff $\Delta(\mathcal{S}_n, \mathcal{S}'_n) = \mathsf{negl}(n)$. We say that two probability ensembles $\mathcal{S}, \mathcal{S}'$ are *computationally indistinguishable*, denoted $\mathcal{S} \overset{c}{\approx} \mathcal{S}'$, if for any PPT distinguisher $|\Pr[\mathcal{D}(\mathcal{S}) = 0] - \Pr[\mathcal{D}(\mathcal{S}') = 0]|$ is negligible. We say an event X in some probability ensemble $\mathcal{S}$ occurs with *overwhelming probability* if the event fails to occur with negligible probability.

A.3 Canetti-Krawczyk model

The authors of [CK02a] present a game-based approach to modeling adaptive adversaries in an authenticated key agreement protocol in the post-specified peer setting. The adversary is permitted to activate new sessions between honest parties, intercept their messages, inject messages, and corrupt them in various ways. We describe in more detail the post-specified peer CK model. Similar definitions exist for pre-specified peers, and can be found in [CK01].

A session activated at party $\mathcal{P}_i$ (the *owner*) is denoted by the tuple $(\mathcal{P}_i, \mathsf{role}, s, d)$, where role is the role (`initiator` or `responder`) of $\mathcal{P}_i$; s is the unique session identifier; and d is the *destination address* of the intended peer, i.e., an abstract identifier that may define a network location or shared address where the intended peer may be found. For brevity, we omit role and d when referring to a session. At the end of a successful session, the party outputs a public triple $(\mathcal{P}_i, s, \mathcal{P}_j)$ where $\mathcal{P}_j$ is the (discovered) peer to the session, and a private session key K. It also erases all its *session state*, i.e., all the ephemeral data used to conduct the protocol. A session that has output its peer-key tuple is called *completed*. A session that has failed is called *aborted* and returns $\perp$.

The model permits the adversary $\mathcal{A}$ to query the following oracles regarding the state of protocol sessions and participating parties.

$\mathsf{NewSession}(\mathcal{P}_i, s, d, \mathsf{role})$ Creates a new session for party $\mathcal{P}_i$ with destination address d. If $\mathsf{role} = \texttt{initiator}$, then $\mathcal{P}_i$ will send an initiating message to the given address. If s is given, the new session is given ID s. If $s = \perp$, then a fresh ID is generated and returned to the caller.

$\mathsf{Send}(\mathcal{P}_i, s, m)$ Sends a message m to $\mathcal{P}_i$ in session s. Returns the response of $\mathcal{P}_i$ after processing the message according to the protocol.

$\mathsf{Corrupt}(\mathcal{P}_i)$ Reveals to $\mathcal{A}$ the long-term keys of $\mathcal{P}_i$, the session keys of all its (unexpired) completed sessions, and the state of all its incomplete sessions.

$\mathsf{RevealKey}(\mathcal{P}_i, s)$ Reveals to $\mathcal{A}$ the session key derived by $\mathcal{P}_i$ in the *completed* session s.

$\mathsf{RevealLtk}(\mathcal{P}_i, s)$ Reveals to $\mathcal{A}$ the long-term keys of $\mathcal{P}_i$.[12]

$\mathsf{RevealState}(\mathcal{P}_i, s)$ Reveals to $\mathcal{A}$ the session state of $\mathcal{P}_i$ in the given *incomplete* session s.

$\mathsf{Expire}(\mathcal{P}_i, s)$ *Expires* the completed session at the given party, i.e., erases the session key of s from $\mathcal{P}_i$'s memory.

$\mathsf{Test}(\mathcal{P}_i, s)$ Can only be called on a completed, unexpired session. Flips a coin b. If $b = 0$, reveals to $\mathcal{A}$ the session key of s held by $\mathcal{P}_i$. If $b = 1$, sends to $\mathcal{A}$ a uniform value from the session key space.

In order to rule out trivial attacks, we must prevent the adversary from corrupting a session's peer and using it to win the test session at the owner. We give the definition of *matching session* from [CK02a].

Definition (Matching session)**.** *Let $(\mathcal{P}_i, s)$ be a completed session with public output $(\mathcal{P}_i, s, \mathcal{P}_j)$. The session $(\mathcal{P}_j, s)$ is called the* matching session *of $(\mathcal{P}_i, s)$ if either*

1. *$(\mathcal{P}_j, s)$ is not completed, or*
2. *$(\mathcal{P}_j, s)$ is completed and its public output is $(\mathcal{P}_j, s, \mathcal{P}_i)$.*

Definition (Session exposure)**.** *A completed session $(\mathcal{P}_i, s)$ with public output $(\mathcal{P}_i, s, \mathcal{P}_j)$ is* exposed *if any of the following holds:*

1. $\mathsf{RevealKey}$ *was called on the session or its matching session (if it exists),*
2. $\mathsf{RevealState}$ *was called on the session or its matching session (if it exists),*
3. $\mathsf{Corrupt}$ *or* $\mathsf{RevealLtk}$ *was called on $\mathcal{P}_i$ or the owner of its matching session (if it exists) at any point.*

[12]This oracle is not actually in the CK model. We show our extended model's equivalence to the base model in Appendix C

We now give the basic notion of security in this model. We will build on this by adding notions of KCIR and full FS.

Definition (Session key (SK) security)**.** *An authenticated key exchange protocol* Π *has session key security if the following hold*

1. Π *is* correct*, i.e., if two uncorrupted parties complete matching sessions, then their session keys are equal with overwhelming probability.*
2. *The advantage of an adversary in distinguishing* $b = 0$ *versus* $b = 1$ *in an unexposed* Test *session is negligible.*

Definition (Key compromise impersonation resistance (KCIR))**.** *An adversary* $\mathcal{A}$ *breaks* SK-security with KCIR *against an AKE protocol iff it wins the SK-security game with the following extra capability:* $\mathcal{A}$ *may learn the long-term key of the owner of the test session at any time (via, e.g.,* RevealLtk *or* Corrupt *outside the lifetime of* s*).*

Definition (Forward secrecy (FS))**.** *An adversary* $\mathcal{A}$ *breaks* SK-security with FS *against an AKE protocol iff it wins the SK-security game with the following extra capabilities: (1)* $\mathcal{A}$ *may learn the long-term key of the owner of the test session after it is expired; (2) similarly, if the test session has a matching session,* $\mathcal{A}$ *may learn the long term key of the owner of that session after it is expired.*

SK-security with KCIR and full FS is the combination of the above three definitions. Note that the combination of capabilities afforded by KCIR and FS gives the attacker the novel ability to get the long-term key of the owner of the test session, expire the session, and call Corrupt on the test session peer.

A.3.1 The id-CK model

The CK model has been extended in prior works to apply to identity-based authenticated key exchange protocols [SE15, HC09, FSXY12]. The changes to CK are minimal:

1. Every party is given a unique identifier id, chosen by the adversary. In protocol outputs and session tuples, id_i replaces $\mathcal{P}_i$.
2. At the beginning of the game, the main keypair $(\mathsf{mpk}, \mathsf{msk})$ is generated. mpk is given to the adversary.
3. There an additional oracle, RevealMsk, which reveals to $\mathcal{A}$ the value of msk. The notion of exposure is extended to include: RevealMsk was called at any point.

Finally, since there is a new type of secret, msk, we extend FS to the id-CK setting.

Definition (Key generation center forward secrecy (KGC-FS))**.** *An adversary* $\mathcal{A}$ *breaks* SK-security with KGC-FS *against an IBKE protocol iff it wins the SK-security game with the following extra capability:* $\mathcal{A}$ *may call* RevealMsk *after both the test session is expired, and its matching session (if it exists) are expired.*

Note that the above definition is equivalent to the ordinary FS, treating RevealMsk as if it were a call to RevealLtk on both the owner of the test session and the owner of its matching session (if it exists).

Finally, we combine our notions of FS:

Definition (Full FS)**.** *An IBKE achieves* full FS *iff it has FS and KGC-FS.*

B Proof of main theorem

We prove the main theorem of the paper, Theorem 1, which claims LATKE UC-realizes $\mathcal{F}_{\mathsf{iPAKE}}$ given an IBKE and a symmetric encryption scheme. The proof consists of multiple game hops, beginning with the real world, i.e., the protocol itself, and ending in the ideal world, i.e., where the simulator may only make calls to $\mathcal{F}_{\mathsf{iPAKE}}$. For brevity we do not write out each game's simulator in full, but we include the final simulator in Figure 7.

We state the proof below for $\text{LATKE}^{\text{post}}$. The argument for $\text{LATKE}^{\text{pre}}$ for pre-specified peer IBKEs is nearly identical. The only difference is that the simulator must use the IDs given in the initial PAKE to activate IBKE sessions in its SK-security reduction. This is a procedural change, and does not affect the soundness of any reduction.

Beyond CK. The below proof references the *id-CK** model as a stand-in for any of the identity-based variants of CK, eCK, CK_{HMQV}, or CK^{+} (adding the RevealLtk oracle as necessary, as explained in Appendix C). It is known that these models all differ [Cre11], but the differences are subtle and are finer than the somewhat coarse UC definitions we target. More specifically, the models differ in their notions of matching sessions (and hence, who is allowed to be compromised and which session keys can be revealed) and permissiveness of attack scenarios.

The differences in matching sessions lie in whether sessions match when their transcripts match, or their ssid's and roles match, or some combination thereof. Our reduction does not corrupt or reveal the session key to sessions that satisfy any of these definitions. We also avoid a correctness issue—whether sessions with matching ssid but non-matching transcripts will output the same key—by simply hashing the transcript into the final key.

The differences in attack permissiveness in these models comes down to the adversary's ability to use short-term state revelation (called "state revelation" in CK-type models, and "ephemeral key revelation" in eCK-type models). Since our reduction never uses short-term state revelation, we are able to use the weakest notion of security, i.e., the least permissive to the adversary, in any of these models. In particular, this means that our notion of key compromise impersonation resistance and full forward secrecy are covered by those in any of these models.

Proof. **Game 0.** Real world

Game 1. $\mathcal{S}$ simulates $\mathcal{F}_{\mathsf{PAKE}}$ and P. $\mathcal{S}$ also simulates $\mathcal{F}_{\mathsf{RO}}$ using hash tables $\mathsf{H}_0, \mathsf{H}_1, \mathsf{H}_2$. This is perfectly indistinguishable from Game 0.

Game 2. We introduce $\mathcal{F}_{\mathsf{iPAKE}}$ and have parties call $\mathcal{F}_{\mathsf{iPAKE}}.\mathsf{StorePwdFile}$ on initialization instead of directly generating its mpk. Instead, a party will generate its mpk lazily via $\mathsf{IBKE.Setup}$ on its first session initiation. After the setup, the simulator will simulate the rest of the session. In this game, the simulator must do two things: (1) consistently respond to corruption queries even for parties that haven't participated in an active session, and (2) simulate honest parties in a session. For an honest party $\mathcal{P}_i$ with password pw_i, we denote its $\mathsf{IBKE.Setup}$ randomness by coins_i. $\mathcal{S}$ will read from and program H_0 in order to make consistent choices for coins_i.

For $\mathsf{StealPwdFile}$ (resp. $\mathsf{Corrupt}$) queries on parties who have not yet begun a session, $\mathcal{S}$ calls $\mathcal{F}_{\mathsf{iPAKE}}.\mathsf{StealPwdFile}$ (resp. $\mathsf{Corrupt}$) and receives $(\mathsf{Stolen}, \mathsf{id}_i, \mathsf{pw}_i)$, where $\mathsf{pw}_i = \perp$ if the password has not yet been guessed. $\mathcal{S}$ must now determine that party's randomness, coins_i. If $\mathsf{pw}_i \neq \perp$, $\mathcal{S}$ sets $\mathsf{coins}_i := \mathsf{H}_0[\mathsf{sid}, \mathsf{pw}_i]$. Otherwise $\mathcal{S}$ performs $\mathcal{F}_{\mathsf{iPAKE}}.\mathsf{OfflineComparePwd}$ queries to see if the stolen password file matches any other known stolen password files. If there are any matches, then coins_i is set to the other party's coins. Otherwise, this a fresh compromise, so $\mathsf{coins}_i \leftarrow\!\!\$\ \{0,1\}^m$. $\mathcal{S}$ then saves a record $(\mathsf{KnownCoins}, \mathcal{P}_i, \mathsf{id}_i, \mathsf{coins}_i)$. Now that coins_i is set, $\mathcal{S}$ may generate the main keypair $\mathsf{msk}_i, \mathsf{mpk}_i$ and extract the user keys $\mathsf{usk}_i, \mathsf{upk}_i$ with respect to id_i. $\mathcal{S}$ returns all these values to the environment.

To maintain consistency of coins, we must also modify how $\mathcal{S}$ simulates H_0 queries. On query $(\mathsf{sid}, \mathsf{pw})$, $\mathcal{S}$ calls $\mathcal{F}_{\mathsf{iPAKE}}.\mathsf{OfflineTestPwd}$ for each compromised party to test if pw produces their stolen password. If there is a match on party $\mathcal{P}_j$, then $\mathcal{S}$ sets $\mathsf{H}_0[\mathsf{sid}, \mathsf{pw}] := \mathsf{coins}_j$, where coins_j comes from the record $(\mathsf{KnownCoins}, \mathcal{P}_j, \cdot, \mathsf{coins}_j)$.

Finally, $\mathcal{S}$ simulates honest parties in the protocol as follows. For a session with an honest party $\mathcal{P}_i$, $\mathcal{S}$ receives mpk_i via its simulation of $\mathcal{F}_{\mathsf{PAKE}}.\mathsf{NewSession}$. From mpk_i, $\mathcal{S}$ can find the corresponding pw_i by testing which pw' yields $(\mathsf{mpk}_i, \cdot) = \mathsf{IBKE.KeyGen}(1^\lambda; H_0[\mathsf{sid}, \mathsf{pw}'])$. This is guaranteed to exist because an honest party in an active session must have called $\mathsf{IBKE.KeyGen}$ on a password. If the user keypair $\mathsf{usk}_i, \mathsf{upk}_i$ has not yet been extracted for the party, $\mathcal{S}$ does so now via $\mathsf{IBKE.Extract}$, and save it for future simulations. $\mathcal{S}$ then compares the mpk provided by both parties in the PAKE session. If the PAKE fails, $\mathcal{S}$ does not need to know the parties' secrets in order to simulate. If the PAKE succeeds, then the mpk values are the same, and if one of them is honest, then $\mathcal{S}$ knows the msk for both, and can thus generate user keypairs for both sides and use them for a complete IBKE simulation $A(\mathsf{mpk}, \mathsf{upk}_A, \mathsf{usk}_A) \Leftrightarrow B(\mathsf{mpk}, \mathsf{upk}_B, \mathsf{usk}_B)$. Since the functionality is symmetric, we assume for simplicity that the first party to $\mathsf{NewSession}$ is the initiator in the IBKE.

This simulation is perfect.

Game 3. Rather than returning the session key directly to the parties, $\mathcal{S}$ now uses $\mathcal{F}_{\mathsf{iPAKE}}.\mathsf{NewKey}$. Since this procedure will only use the provided key if the session is marked `compromised`, the simulator must call the appropriate compromising functions, $\mathsf{Impersonate}$ and $\mathsf{OnlineTestPwd}$, when possible.

We define the simulator to mark a compromise at multiple points. Firstly, any time an mpk is provided to the protocol by a corrupted party (in the next step uncorrupted parties will not provide mpk at all), $\mathcal{S}$ checks if there is a pw such that $(\mathsf{mpk}, \cdot) = \mathsf{IBKE.KeyGen}(1^\lambda; \mathsf{coins})$, where $\mathsf{coins} = \mathsf{H}_0[\mathsf{sid}, \mathsf{pw}]$. If there is a match, $\mathcal{S}$ calls $\mathcal{F}_{\mathsf{iPAKE}}.\mathsf{OnlineTestPwd}$ with that password. This is done in the simulation of $\mathcal{F}_{\mathsf{PAKE}}.\mathsf{TestPwd}$ and $\mathcal{F}_{\mathsf{PAKE}}.\mathsf{NewKey}$. Secondly, if an party uses a compromised mpk, i.e., one that is the result of $\mathsf{IBKE.KeyGen}(1^\lambda; \mathsf{coins}_k)$ for some recorded coins_k belonging to party $\mathcal{P}_k$, and the session is tampered with by the adversary, then $\mathcal{S}$ calls $\mathcal{F}_{\mathsf{iPAKE}}.\mathsf{Impersonate}$ on the corresponding party as $\mathcal{P}_k$.[13] Similarly, if both parties are compromised, then $\mathsf{Impersonate}$ is called on both. $\mathsf{Impersonate}$ is also done in the simulation of $\mathcal{F}_{\mathsf{PAKE}}.\mathsf{TestPwd}$ and $\mathcal{F}_{\mathsf{PAKE}}.\mathsf{NewKey}$. Finally, if an honest party $\mathcal{P}_i$ is corrupted during a session with another honest party, $\mathcal{S}$ calls $\mathcal{F}_{\mathsf{PAKE}}.\mathsf{Impersonate}$ as $\mathcal{P}_i$, "impersonating" itself.

Game 2 $\overset{c}{\approx}$ Game 3. We argue that this is indistinguishable from the last game, assuming IBKE is SK-secure with KCIR and full FS.

Let $\mathcal{Z}$ be an environment that can distinguish between the two games. We will use $\mathcal{Z}$ to construct an adversary $\mathcal{A}$ against the SK-security game with KCIR and full FS. We will show the reduction for a sequence of hybrids. In hybrid i, all sessions $j \geq i$ return the key directly to the user, and all sessions $j < i$ call $\mathcal{F}_{\mathsf{iPAKE}}.\mathsf{NewKey}$. Hybrid 0 is game 2, and hybrid N is game 3, where N is the (polynomial) number of sessions in the experiment.

Note that, between these two games, the only time NewKey behaves differently (i.e., sending a random key) is when, at the end of the protocol execution, both parties are uncorrupted and at most one is compromised.[14] In addition, NewKey is only different when mpk doesn't come from a known password (else $\mathcal{S}$ runs $\mathsf{OnlineTestPwd}$ and wins). Finally, NewKey is only different when the initial PAKE succeeds, i.e., both parties submitted the same mpk (otherwise both parties output randomness). Thus, for hybrid i, we may assume wlog that the session in question ends with both parties uncorrupted, at most one party compromised, $\mathsf{H}_0[\mathsf{sid}, \mathsf{pw}]$ belonging to the mpk not queried, and both parties submitted the same mpk to the PAKE. We can safely assume this because, outside of this scenario, the simulator's success probability is 1.

We reduce hybrid i from id-CK* now, defining an adversary $\mathcal{A}$. Let mpk^* be the main public key provided by the id-CK* challenger. Let session i be a session between parties $\mathcal{P}$ and $\mathcal{P}'$. This will be the Test session. We program the mpk of both $\mathcal{P}$ and $\mathcal{P}'$ to be mpk^*. We use the id-CK* Send oracle when simulating the sending of messages in every session using mpk^* as $\mathcal{P}$ and $\mathcal{P}'$, $\mathsf{CK.RevealLtk}$ for $\mathsf{StealPwdFile}$ queries, $\mathsf{CK.Corrupt}$ for $\mathsf{Corrupt}$ queries (by hypothesis, on parties

[13]The tampering condition is to handle the following scenario. If Alice is compromised and Mallory breaks the PAKE session between Alice and Bob, but otherwise does not tamper with any messages, then the session should succeed. That is, NewKey must produce the same key for both parties. If Mallory modifies or injects messages, though, then Bob's output must be consistent with Mallory's view, so $\mathcal{S}$ must $\mathsf{Impersonate}$ to Bob. If Bob is not compromised, then this breaks the symmetry of the keys. The final transcript hash helps us break this symmetry.

[14]Note that this still permits an adversary to participate in the encrypted portion of the protocol, assuming they passed the PAKE using a stolen mpk.

other than $\mathcal{P}$ and $\mathcal{P}'$), and $\mathsf{RevealKey}$ to determine the final key of all other sessions. If the adversary did not modify the IBKE messages during a session (either by knowing the PAKE key and modifying the messages, or not knowing the PAKE key and modifying the ciphertext), then the iPAKE output key for *both* parties is the one given by Test. This corresponds to the condition that matching completed sessions between honest parties must produce the same key, and non-matching sessions overwhelmingly do not produce the same key (otherwise an adversary wins by simply calling $\mathsf{RevealKey}$ on the non-matching session). We return these as described above, based on the session number. For the test session (which is clean by hypothesis), we simply return the challenge to $\mathcal{P}$ and $\mathcal{P}'$. Clearly, $b = 0$ is hybrid $i - 1$ and $b = 1$ is hybrid i. $\mathcal{A}$ returns whatever $\mathcal{Z}$ guesses at the end of the UC game. Thus the advantage is precisely the SK-security with KCIR and full FS advantage.

Game 4. We replace AE with the simulator $\mathsf{S}_{\mathsf{cca}}$ whenever a PAKE completed and at most one of the parties is compromised. Recall, rather than using just the party $\mathcal{P}_i$ as a user identifier for $\mathsf{S}_{\mathsf{cca}}$, we use $(\mathcal{P}_i, \mathsf{ssid}, \mathsf{step})$, where step refers to the current step in the protocol. This corresponds to the fact that the real-world protocol uses a unique key for every message.

Recall the simulator knows the msk and usk of any party in a session where at least one compromise has occurred. Thus, it knows the output of the PAKE in these scenarios and does not need to use any indistinguishability property. Similarly, the simulator learns the value of the PAKE if $\mathcal{F}_{\mathsf{PAKE}}.\mathsf{TestPwd}$ is called on the session. The only case we must handle is where, during the EUE phase of a session with two honest parties and an uncompromised PAKE session, one party is corrupted.

When $\mathsf{Corrupt}$ is called on $\mathcal{P}_i$, $\mathcal{S}$ uses the long term keys to check if the PAKE failed or if any messages were modified by the adversary. Either of these events imply a decryption error in the real world. Thus, the simulator creates a valid IBKE transcript between $\mathcal{P}_i$ and its peer $\mathcal{P}_j$, up until the point of failure (if at all), and then, per EUE, pads the rest of the transcript with zeros. The simulator selects a random chain key K_{ch} and programs the decryption key $k = \mathsf{AE.Exp}(m, c)$, where c is the last ciphertext to be received by $\mathcal{P}_i$ in the protocol and m last message to be received by $\mathcal{P}_i$ in the fabricated transcript.

After corruption, $\mathcal{S}$ switches to using AE and the revealed key. If the corruption occurred after the last message was sent, then $\mathcal{S}$ programs $\mathsf{H}_1(K', \mathsf{tr})$ to be the random session key previously chosen.

In order to make the keys consistent, we must also program H_2 so that $\mathsf{H}_2(\mathsf{sid}, \mathsf{ssid}, 1\|K_{\mathsf{ch}}^{(i)}) = (K_{\mathsf{ch}}^{(i+1)}, k_{i+1})$ for every chain key $K_{\mathsf{ch}}^{(i)}$ and exposed encryption key k_i, and $\mathsf{H}_2(\mathsf{sid}, \mathsf{ssid}, 0\|K) = K_{\mathsf{ch}}^{0}$ for the initial key.

Game 3 $\overset{c}{\approx}$ Game 4. We reduce this hop to the SIM*-AC-CCA security of AE. The only bad situation to handle is if H_2 has already been called on one of the chain keys. Since the PAKE keys are unknown to the adversary by hypothesis, and all keys are defined in a random oracle hash chain, this probability is negligible.

We can show this game reduces to SIM*-AC-CCA using a sequence of hybrids.

We define hybrid i as the world where every iPAKE subsession $\leq i$ is defined without the simulator, and each $> i$ is. Each game hop goes from defining a ciphertext $c \coloneqq \mathsf{AE.Enc}_k(m)$ for a known key k and message m, to defining $c \coloneqq \mathsf{S_{cca}.Enc}$ and later generating k as the opening of c to m. Further, all tampered ciphertexts are assumed by the simulator to trigger a decryption error. This is precisely the SIM*-AC-CCA security game. Thus, the advantage of any adversary in the larger game hop is bounded above by $N \cdot \mathsf{Adv}_{\mathcal{A}}^{\mathrm{sim*-ac-cca}}$, where N is the number of sessions in the game.

Game 5. We now remove the ability of $\mathcal{S}$ to see the PAKE keys for sessions with participants who are honest up to and including $\mathcal{F}_{\mathsf{PAKE}}.\mathsf{NewKey}$. On activation, each party calls $\mathcal{F}_{\mathsf{iPAKE}}.\mathsf{NewSession}$ instead of directly calling $\mathcal{F}_{\mathsf{PAKE}}.\mathsf{NewSession}$. Using $\mathsf{NewSession}$ means that $\mathsf{OnlineTestPwd}$ can return meaningful answers.

Since the specific value of the PAKE keys no longer matters (since $\mathsf{S_{cca}}$ allows $\mathcal{S}$ to generate the appropriate hash chain key), the thing $\mathcal{S}$ loses in this hop is knowing whether a PAKE session succeeded, i.e., both parties got the same key. Specifically, $\mathcal{S}$ must know this for sessions with at least one compromised party in order for the above steps to work. Thus, it will suffice to define a way for $\mathcal{S}$ to determine the success of any such PAKE session.

We define $\mathcal{S}$ to check every possible value to determine whether the PAKE between $\mathcal{P}_i$ and $\mathcal{P}_j$ succeeded. If the parties are both corrupted, then they provided their own mpk values, and those can be compared directly. If only one of them is corrupted, then that supplied mpk value is checked against known compromised users (use $\mathsf{OnlineComparePwd}$) and compromised passwords (use $\mathsf{OnlineTestPwd}$). Finally, if one party is compromised and not corrupted then its mpk is checked against the other's (use $\mathsf{OnlineTestPwd}$). This new simulator covers every edge case using just $\mathcal{F}_{\mathsf{iPAKE}}^{\mathrm{ocw}}$ functionality, and so this hop is perfectly indistinguishable from the last. This completes our proof. $\square$

```
On corruption query $(\mathsf{StealPwdFile}, \mathsf{sid}, \mathcal{P}_i)$ from $\mathcal{Z}$
  send $(\mathsf{StealPwdFile}, \mathsf{sid}, \mathcal{P}_i)$ to $\mathcal{F}_{\mathsf{iPAKE}}$
  if received "no record" :
    send "no record" to $\mathcal{Z}$ and exit
  elsereceived $(\mathsf{Stolen}, \mathsf{sid}, \mathsf{id}_i, \mathsf{pw}_i)$
  if $\mathsf{pw}_i \neq \bot, \mathsf{coins}_i := \mathsf{H}_0[\mathsf{sid}, \mathsf{pw}_i]$
  else : $\mathsf{coins}_i \leftarrow\!\!\$ \{0,1\}^m$
  for each record $(\mathsf{KnownCoins}, \mathcal{P}_j, \cdot, \mathsf{coins}_j)$ :
    send $(\mathsf{OfflineComparePwd}, \mathcal{P}_i, \mathcal{P}_j)$ to $\mathcal{F}_{\mathsf{iPAKE}}$
    if received "match", $\mathsf{coins}_i := \mathsf{coins}_j$
  record $(\mathsf{KnownCoins}, \mathcal{P}, \mathsf{id}, \mathsf{coins}_i)$
  $(\mathsf{msk}_i, \mathsf{mpk}_i) := \mathsf{IBKE.Setup}(1^\lambda; \mathsf{coins}_i)$
  $(\mathsf{usk}_i, \mathsf{upk}_i) \leftarrow \mathsf{IBKE.Extract}_{\mathsf{msk}_i}(\mathsf{id}_i)$
  $\mathsf{ibkeStates}[\mathsf{sid}, \cdot, \mathcal{P}_i, 0] := (\mathtt{begin}, \mathsf{msk}_i, \mathsf{usk}_i)$
  Mark $\mathcal{P}_i$ compromised
  for each $(\mathsf{Session}, \mathsf{sid}, \mathsf{ssid}, \mathcal{P}_i, \mathcal{P}_j, \cdot, \mathsf{role})$
   marked pakecompleted and not complete :
    $\mathsf{EueCatchup}(\mathsf{sid}, \mathsf{ssid})$
  record $(\mathsf{Pwfile}, \mathsf{sid}, \mathcal{P}_i, \mathsf{mpk}_i, \mathsf{upk}_i, \mathsf{usk}_i)$
  send $(\mathsf{mpk}_i, \mathsf{upk}_i, \mathsf{usk}_i)$ to $\mathcal{Z}$

On corruption query $(\mathsf{Corrupt}, \mathcal{P}_i)$ from $\mathcal{Z}$
  $(\mathsf{mpk}_i, \mathsf{upk}_i, \mathsf{usk}_i) := \mathcal{S}.\mathsf{StealPwdFile}(\mathcal{P}_i)$
  $\mathsf{curKeys} := \{\}$
  for each $(\mathsf{Session}, \mathsf{sid}, \mathsf{ssid}, \mathcal{P}_i, \cdot, \cdot, \cdot)$not completed :
    $(K_{\mathsf{ch}}, k) := \mathsf{eueKeys}[\mathsf{sid}, \mathsf{ssid}, \mathcal{P}_i]$
    $\mathsf{curKeys.append}((\mathsf{sid}, \mathsf{ssid}, K_{\mathsf{ch}}, k))$
  Mark $\mathcal{P}$ corrupted and compromised
  send $(\mathsf{mpk}_i, \mathsf{upk}_i, \mathsf{usk}_i, \mathsf{curKeys})$ to $\mathcal{Z}$

On $(\mathsf{TestPwd}, \mathsf{sid}, \mathsf{ssid}, \mathcal{P}_i, \mathsf{mpk})$ from $\mathcal{Z}$ to $\mathcal{F}_{\mathsf{PAKE}}$
  retrieve $(\mathsf{Session}, \mathsf{sid}, \mathsf{ssid}, \mathcal{P}_i, \mathcal{P}_j, \cdot, \mathsf{mpk}_i)$
  retrieve $(\mathsf{Session}, \mathsf{sid}, \mathsf{ssid}, \mathcal{P}_j, \mathcal{P}_i, \cdot, \mathsf{mpk}_j, \cdot)$
  $\mathsf{succ} := \mathtt{false}$
  if $\mathsf{mpk} = \mathsf{mpk}_i \neq \bot$ :
    $\mathsf{succ} := \mathtt{true}$
  else :
    if received "correct" : $\mathsf{succ} := \mathtt{true}$
  if succ :
    Mark session compromised
    return "correct"
  else :
    Mark session interrupted
    return "wrong"

On $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{ssid}, \mathcal{P}_j, \mathsf{mpk})$ from corrupt $\mathcal{P}_i$ to $\mathcal{F}_{\mathsf{PAKE}}$
  if $\nexists$ a record $(\mathsf{Session}, \mathsf{sid}, \mathsf{ssid}, \mathcal{P}_i, \mathcal{P}_j, \cdot, \cdot, \cdot)$
    $\mathsf{role} :=$ if $\nexists(\mathsf{Session}, \mathsf{sid}, \mathsf{ssid}, \mathcal{P}_j, \mathcal{P}_i, \cdots)$ : init else resp
    record $(\mathsf{Session}, \mathsf{ssid}, \mathcal{P}_i, \mathcal{P}_j, \mathsf{id} = \bot, \mathsf{mpk}, \mathsf{role})$
      marked fresh

On $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{ssid}, \mathcal{P}_i, \mathcal{P}_j, \mathsf{id}_i, \mathsf{role})$ from $\mathcal{F}_{\mathsf{iPAKE}}$
  $\mathsf{role} :=$ if $\nexists(\mathsf{Session}, \mathsf{sid}, \mathsf{ssid}, \mathcal{P}_j, \mathcal{P}_i, \cdots)$ : init else resp
  record $(\mathsf{Session}, \mathsf{ssid}, \mathcal{P}_i, \mathcal{P}_j, \mathsf{id}_i, \mathsf{mpk} = \bot, \mathsf{role})$
    marked fresh

On $(\mathsf{NewKey}, \mathsf{sid}, \mathsf{ssid}, \mathcal{P}_i, K')$ from $\mathcal{Z}$ to $\mathcal{F}_{\mathsf{PAKE}}$
  retrieve $(\mathsf{Session}, \mathsf{ssid}, \mathcal{P}_i, \mathcal{P}_j, \cdot, \mathsf{mpk}_i, \mathsf{role})$
  retrieve $(\mathsf{Session}, \mathsf{ssid}, \mathcal{P}_j, \mathcal{P}_i, \cdot, \mathsf{mpk}_j, \cdot)$
    not marked pakecompleted
  if session is compromised :
    $K_i := K'$
  elif session is fresh and
   $\exists(\mathsf{FreshKey}, \mathsf{sid}, \mathsf{ssid}, \mathcal{P}_j, K_j)$ :
    $\mathsf{iseq} := \mathsf{SessMpkEq}(\mathsf{ssid})$
    if $\mathsf{iseq} = \mathtt{true}$ : $K_i := K_j$
    elif $\mathsf{iseq} = \mathtt{false}$ : $K_i \leftarrow\!\!\$ \mathcal{K}$
    else : $K_i := \mathtt{indeterminate}$
  else : $K_i \leftarrow\!\!\$ \mathcal{K}$
  if fresh : record $(\mathsf{FreshKey}, \mathsf{sid}, \mathsf{ssid}, \mathcal{P}_i, K_i)$
  Mark session pakecompleted
  if $K_i \neq \mathtt{indeterminate}$ :
    record $(\mathsf{ChoseKey}, \mathsf{ssid}, \mathcal{P}_i, K_i)$
    if both pakecompleted :
      retrieve $(\mathsf{ChoseKey}, \mathsf{ssid}, \mathcal{P}_j, K_j)$
      $\mathsf{eueKeys}[\mathsf{sid}, \mathsf{ssid}, \mathcal{P}_i] := \mathsf{H}_2(\mathsf{sid}, \mathsf{ssid}, K_i)$
      $\mathsf{eueKeys}[\mathsf{sid}, \mathsf{ssid}, \mathcal{P}_j] := \mathsf{H}_2(\mathsf{sid}, \mathsf{ssid}, K_j)$
  if $\mathcal{P}_i$ corrupted : send $(\mathsf{ssid}, K_i)$ to $\mathcal{P}_i$

On $(\mathsf{Hash}, \mathsf{sid}, (i, x))$ from $\mathcal{Z}$ to $\mathcal{F}_{\mathsf{RO}}$
  if $i = 0$, for each party $\mathcal{P}$ :     // program keygen coins
    $\mathsf{pw} := x$
    send $(\mathsf{OfflineTestPwd}, \mathsf{ssid}, \mathcal{P}, \mathsf{pw})$ to $\mathcal{F}_{\mathsf{iPAKE}}$
    if received "correct" :     // "correct" $\implies$ compromised
      retrieve $(\mathsf{Compromised}, \mathcal{P}, \cdot, \mathsf{coins})$
      $\mathsf{H}_0[\mathsf{sid}, \mathsf{pw}] := \mathsf{coins}$
  return $\mathsf{H}_i[\mathsf{sid}, x]$
```

Figure 7: The final UC simulator $\mathcal{S}$ for the LATKE$^{\text{post}}$ protocol. $\mathcal{S}$ maintains hash tables, $\mathsf{H}_0, \mathsf{H}_1, \mathsf{H}_2$. Hash queries $\mathsf{H}_i[x]$ are defined to be uniform and consistent when not otherwise specified.

```
On msg $(\mathcal{P}_j \to \mathcal{P}_i, c, \mathsf{sid}, \mathsf{ssid})$
  assert all $(\mathsf{Session}, \mathsf{sid}, \mathsf{ssid}, \cdots)$ are pakecompleted
  retrieve $(\mathsf{Session}, \mathsf{sid}, \mathsf{ssid}, \mathcal{P}_i, \mathcal{P}_j, \cdot, \mathsf{id}_i, \mathsf{role})$
  $\mathsf{step} := |\mathsf{sentMsgs}[\mathsf{sid}, \mathsf{ssid}, \mathcal{P}_i, \cdot]|$
  $\mathsf{recvdMsgs}[\mathsf{sid}, \mathsf{ssid}, \mathcal{P}_i, \mathsf{step}] := c$
  if $\exists(\mathsf{KnownCoins}, \mathcal{P}_i, \mathsf{id}_i, \mathsf{coins}_i)$ :
    $(k, K_{\mathsf{ch}}) := \mathsf{eueKeys}[\mathsf{sid}, \mathsf{ssid}, \mathcal{P}_i]$
    $\mathsf{st} := \mathsf{ibkeStates}[\mathsf{sid}, \mathsf{ssid}, \mathcal{P}_i]$
    $\mathsf{inCt} := \mathsf{recvdMsgs}[\mathsf{sid}, \mathsf{ssid}, \mathcal{P}_i, \mathsf{step} - 1]$ or begin
    if $\mathsf{st} \neq \bot$ :
      $\mathsf{inMsg} := \mathsf{AE.Dec}^{\mathsf{P}}_k(\mathsf{inCt})$
      if $\mathsf{inMsg} \neq \bot$ :
        $(\mathsf{outMsg}, \mathsf{st}') := \mathsf{IBKE.next}(\mathsf{st}, \mathsf{inMsg})$
        if $\mathsf{st}' = (\mathsf{Done}, \cdot, \cdot)$ :
          $\mathsf{FinishIbke}(\mathsf{sid}, \mathsf{ssid}, \mathsf{st}')$
          return $\bot$
        $\mathsf{ibkeStates}[\mathsf{sid}, \mathsf{ssid}, \mathcal{P}_i] := \mathsf{st}'$
      else : $\mathsf{outMsg} := 0^{\mathsf{msgSize}_{\mathsf{step}}}$
    else : $\mathsf{outMsg} := 0^{\mathsf{msgSize}_{\mathsf{step}}}$
    $\mathsf{outCt} := \mathsf{AE.Enc}^{\mathsf{P}}_k(\mathsf{outMsg})$
    $(k', K'_{\mathsf{ch}}) := \mathsf{H}_1(K_{\mathsf{ch}})$
    $\mathsf{eueKeys}[\mathsf{sid}, \mathsf{ssid}, \mathcal{P}_i] := (k', K'_{\mathsf{ch}})$
  else :
    $\mathsf{outCt} \leftarrow\!\!\$ \{0,1\}^{\mathsf{msgSize}_{\mathsf{step}}+\tau}$
  send $(\mathcal{P}_i \to \mathcal{P}_j, \mathsf{ssid}, \mathsf{outCt})$
  $\mathsf{sentMsgs}[\mathsf{sid}, \mathsf{ssid}, \mathcal{P}_i, \mathsf{step}] := \mathsf{outCt}$
```

```
Procedure $\mathsf{EueCatchup}(\mathsf{sid}, \mathsf{ssid})$
    // assume wlog $i$ is the compromised party
  retrieve $(\mathsf{Session}, \mathsf{sid}, \mathsf{ssid}, \mathcal{P}_i, \mathcal{P}_j, \mathsf{id}_i, \mathsf{mpk}_i, \mathsf{role})$
  retrieve $(\mathsf{Session}, \mathsf{sid}, \mathsf{ssid}, \mathcal{P}_j, \mathcal{P}_i, \cdot, \mathsf{mpk}_j, \cdot)$
  $\mathsf{steps} = |\mathsf{sentMsgs}[\mathsf{ssid}, \mathcal{P}_i, \cdot]|$
  if $\mathsf{SessMpkEq}(\mathsf{ssid})$ :
    if $\exists(\mathsf{FreshKey}, \mathsf{sid}, \mathsf{ssid}, \mathcal{P}_i, K_i)$ : $K := K_i$
    else : $K \leftarrow\!\!\$ \{0,1\}^{\lambda}$
    $K_{\mathsf{ch}} := \mathsf{H}_2(\mathsf{sid}, \mathsf{ssid}, 0 \| K)$
    $\mathsf{eueKeys}[\mathsf{sid}, \mathsf{ssid}, \mathcal{P}_i] := \mathsf{H}_2(\mathsf{sid}, \mathsf{ssid}, 1 \| K_{\mathsf{ch}})$
    $\mathsf{ibkeStates}[\mathsf{sid}, \mathsf{ssid}, \mathcal{P}_j, \mathsf{step}] := (\mathtt{begin}, \mathsf{msk}, \mathsf{usk}_j)$
  $\mathsf{outMsg} := \emptyset$
  for step in $0 \ldots \mathsf{steps}$ :
    $\mathsf{st} := \mathsf{ibkeStates}[\mathsf{sid}, \mathsf{ssid}, \mathcal{P}_k, \mathsf{step}]$ or $\bot$
    if step is even :
      $\ell :=$ if $\mathsf{role} = \mathtt{init}$ : $i$ else $j$
    else :
      $\ell :=$ if $\mathsf{role} = \mathtt{init}$ : $j$ else $i$
    if $\mathsf{st} = \bot$ or
     $\mathsf{recvdMsgs}[\mathsf{sid}, \mathsf{ssid}, \mathcal{P}_k, \mathsf{step} - 1]$
      $\neq \mathsf{sentMsgs}[\mathsf{sid}, \mathsf{ssid}, \mathcal{P}_k, \mathsf{step} - 1]$ :
      $\mathsf{outMsg} := 0^{\mathsf{msgSize}_{\mathsf{step}}}$
      $\mathsf{st}' := \bot$
    else :
      $\mathsf{outCt} := \mathsf{sentMsgs}[\mathsf{ssid}, \mathcal{P}_k, \mathsf{step}]$
      $(\mathsf{outMsg}, \mathsf{st}') := \mathsf{IBKE.next}(\mathsf{st})$
    $k := \mathsf{S}_{\mathsf{cca}}.\mathsf{Exp}((\mathsf{sid}, \mathsf{ssid}, \mathcal{P}_k, \mathsf{step}), (\mathsf{outMsg}, \mathsf{outCt}))$
    $K'_{\mathsf{ch}} \leftarrow\!\!\$ \{0,1\}^{\lambda}$
    $\mathsf{H}_2[\mathsf{sid}, \mathsf{ssid}, 1 \| K_{\mathsf{ch}}] := (K'_{\mathsf{ch}}, k)$
    $\mathsf{ibkeStates}[\mathsf{sid}, \mathsf{ssid}, \mathcal{P}_k, \mathsf{step}] := \mathsf{st}'$
    $\mathsf{eueKeys}[\mathsf{sid}, \mathsf{ssid}, \mathcal{P}_k] := (K'_{\mathsf{ch}}, k)$
```

```
Procedure $\mathsf{FinishIbke}(\mathsf{sid}, \mathsf{ssid}, \mathcal{P}_i, \mathsf{st})$
  retrieve $(\mathsf{Session}, \mathsf{sid}, \mathsf{ssid}, \mathcal{P}_i, \cdot, \cdot, \mathsf{id}_i, \cdot)$
  $(\mathsf{Done}, K, \mathsf{id}') = \mathsf{st}$
  $\mathsf{sessTampered} :=$
   $\mathsf{sentMsgs}[\mathsf{sid}, \mathsf{ssid}, \cdot, \cdot] \neq \mathsf{recvdMsgs}[\mathsf{sid}, \mathsf{ssid}, \cdot, \cdot]$
  if $\mathsf{id}' \neq \mathsf{id}_i$ or sessTampered :
    if id is compromised
      send $(\mathsf{Impersonate}, \mathsf{sid}, \mathsf{ssid}, \mathcal{P}_i, \mathcal{P}_{\mathsf{id}})$
    elif $\exists \mathsf{sid}', \mathsf{pw}$ s.t. $\mathsf{mpk}_i = \mathsf{IBKE.Kg}(1^{\lambda}; \mathsf{H}_0(\mathsf{sid}', \mathsf{pw}))$ :
      send $(\mathsf{OnlineTestPwd}, \mathsf{sid}, \mathsf{ssid}, \mathcal{P}_i, \mathsf{pw})$
    else : abort
  send $(\mathsf{NewKey}, \mathsf{sid}, \mathsf{ssid}, \mathcal{P}_i, \mathsf{id}', K')$

Procedure $\mathsf{SessMpkEq}(\mathsf{sid}, \mathsf{ssid})$
  retrieve $(\mathsf{Session}, \mathsf{sid}, \mathsf{ssid}, \mathcal{P}_i, \mathcal{P}_j, \mathsf{id}_i, \mathsf{mpk}_i, \mathsf{role})$
  retrieve $(\mathsf{Session}, \mathsf{sid}, \mathsf{ssid}, \mathcal{P}_j, \mathcal{P}_i, \cdot, \mathsf{mpk}_j, \cdot)$
  if $\mathsf{mpk}_i \neq \bot \wedge \mathsf{mpk}_j \neq \bot$ :
    return $\mathsf{mpk}_i = \mathsf{mpk}_j$
  $\mathsf{out} := \mathtt{indeterminate}$
  $\mathsf{coins} := \bot$
  if $\exists \ell$ s.t. $\mathsf{mpk}_\ell \neq \bot$ :
    $\hat{\ell} := i$ if $\ell = j$ else $j$
    if $\mathsf{mpk}_\ell$ has owner $\mathcal{P}_\ell$ :
      send $(\mathsf{OnlineComparePwd}, \mathsf{sid}, \mathsf{ssid}, \mathcal{P}_{\hat{\ell}}, \mathcal{P}_\ell)$
      $\mathsf{out} := \mathsf{resp} =$ "correct" :
      if $\mathcal{P}_{\hat{\ell}}$ honest :
        $\mathsf{coins} :=$ coins of $\mathsf{mpk}_\ell$
    elif $\mathsf{mpk}_\ell$ is derived from $\mathsf{pw}_\ell$ :
      if $\mathcal{P}_{\hat{\ell}}$ honest :
        $\mathsf{coins} :=$ coins of $\mathsf{pw}_\ell$
      send $(\mathsf{OnlineTestPwd}, \mathsf{sid}, \mathsf{ssid}, \mathcal{P}_{\hat{\ell}}, \mathsf{pw}_\ell)$
      $\mathsf{out} := \mathsf{resp} =$ "correct" :
  elif $\exists \ell$ s.t. $\mathcal{P}_\ell$ compromised :
    $\hat{\ell} := i$ if $\ell = j$ else $j$
    if $\mathcal{P}_{\hat{\ell}}$ honest :
      $\mathsf{coins} :=$ coins of $\mathsf{mpk}_\ell$
    send $(\mathsf{OnlineComparePwd}, \mathsf{sid}, \mathsf{ssid}, \mathcal{P}_{\hat{\ell}}, \mathcal{P}_\ell)$
    $\mathsf{out} := \mathsf{resp} =$ "correct"
  if $\mathsf{out} = \mathtt{true}$ :
    record $(\mathsf{KnownCoins}, \mathcal{P}, \mathsf{id}_i, \mathsf{coins})$
  return out
```

Figure 8: (cont.) the LATKE$^{\text{post}}$ UC simulator

C RevealLtk in the CK model

In our presentation of the CK model in Appendix A.3, we include the oracle $\mathsf{RevealLtk}$, which, unlike $\mathsf{Corrupt}$, reveals the long-term key of the given party without revealing any internal state. While $\mathsf{RevealLtk}$ is included in the extended Canetti-Krawczyk model (eCK) [LLM07], it does not appear in the original CK or CK_{HMQV} models [CK01, CK02a, Kra05], which Theorem 1 claims compatibility with.

We also note that the SK-with-$\mathsf{RevealLtk}$ game was implicitly used in the security proof of CHIP in version 3.0 of the CHIP paper [CNPR22]. Thus, this section is not only necessary for LATKE, but also fills a gap in the security proof of CHIP.

We claim that this new model is equivalent to the base model. An adversary against the SK-without-$\mathsf{RevealLtk}$ game is trivially an adversary against the SK-with-$\mathsf{RevealLtk}$ game. So it only remains to show the converse. The theorem applies to protocols in both the CK or CK_{HMQV} model.

Theorem 2. *Let* Π *be an authenticated key exchange protocol . If* $\mathcal{B}$ *is an adversary with advantage* ϵ *against the SK-security game with access to the* $\mathsf{RevealLtk}$ *oracle, then there exists an adversary* $\mathcal{A}$ *against the ordinary SK-security game with advantage at least* $\epsilon/4$. *This also applies to SK-security with KCIR, FS, and KCIR+FS.*

Proof. Let $N = \mathsf{poly}(\lambda)$ denote the number of parties that $\mathcal{B}$ initializes in its game. We define $\mathcal{A}$ as follows. At the very beginning of the SK-security game, $\mathcal{A}$ selects $N/2$ parties at random and calls $\mathsf{Corrupt}$ on them, thus receiving all their long term keys. Let C denote the set of the initially corrupted parties, and U the set of initially uncorrupted parties. Let sk_i denote the long-term key of party $\mathcal{P}_i$.

Then, $\mathcal{A}$ runs $\mathcal{B}$. For every oracle query $\mathcal{B}$ makes, $\mathcal{A}$ makes the same query (if it exists), including when choosing the test session. When $\mathcal{B}$ makes a query of the form $\mathsf{RevealLtk}(\mathcal{P}_i)$, then $\mathcal{A}$ (1) checks if it has stored sk_i and, if so, returns it; or (2) calls $\mathsf{Corrupt}$ on $\mathcal{P}_i$, saves the long-term key as sk_i, and returns sk_i. For its test session, $\mathcal{A}$ outputs whichever bit $\mathcal{B}$ outputs. If any of the previous instructions lead $\mathcal{A}$ to expose the test session, a *failsafe* will trigger and $\mathcal{A}$ will instead respond to the test session with a random bit.

It will suffice to consider a specific condition E under which $\mathcal{A}$ is guaranteed to win its game when $\mathcal{B}$ wins its game, i.e.,

$$\Pr[\mathcal{A} \text{ wins} \mid \mathsf{E}] = \Pr[\mathcal{B} \text{ wins}].$$

In other words, when E holds, then $\mathcal{A}$'s test session is unexposed.

In all of the SK-security variants, $\mathsf{RevealLtk}$ and $\mathsf{Corrupt}$ are treated equally in terms of exposure except *during* the test session (i.e., after it is initiated and before it is expired). In the KCIR variant, the adversary is permitted to call $\mathsf{RevealLtk}$ on the owner $\mathcal{P}_i$ of the test session during the test session . If $\mathcal{B}$ does this, then $\mathcal{A}$ may have to call $\mathsf{Corrupt}$ on the owner, exposing the session. Thus, $\mathcal{A}$'s test session remains unexposed if both

1. $\mathcal{P}_i \in C$ (so $\mathcal{A}$ can respond with sk_i), and
2. the peer of the session $\mathcal{P}_j$ (if it exists) is in U (so at least one party is uncorrupted).

Let E be the event in which both statements are true. It is clear that the above equality holds with respect to E in every SK-security variant. Since both (1) and (2) occur independently and with even odds, $\Pr[\mathsf{E}] = 1/4$.

When E does not occur, then $\mathcal{A}$'s failsafe may trigger. We may lower bound $\Pr[\mathcal{A} \text{ wins} \mid \neg\mathsf{E}] \geq 1/2$.

Thus, the advantage of $\mathcal{A}$ is

$$\begin{aligned}
\left|\Pr[\mathcal{A} \text{ wins}] - \frac{1}{2}\right| &= \left|\Pr[\mathcal{A} \text{ wins} \mid \mathsf{E}] \Pr[\mathsf{E}] + \Pr[\mathcal{A} \text{ wins} \mid \neg\mathsf{E}] \Pr[\neg\mathsf{E}] - \frac{1}{2}\right| \\
&\geq \left|\frac{\Pr[\mathcal{B} \text{ wins}]}{4} - \frac{1}{8}\right| \\
&= \frac{1}{4}\left|\Pr[\mathcal{B} \text{ wins}] - \frac{1}{2}\right| \\
&= \epsilon/4
\end{aligned}$$

□