Credential Authenticated Identification and Key Exchange

Jan Camenisch[1], Nathalie Casati[1], Thomas Gross[1], and Victor Shoup[2]

[1] IBM Research, work funded by the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement no. 216483
[2] NYU, work done while visiting IBM Research, supported by NSF grant CNS-0716690

Abstract. This paper initiates a study of two-party identification and key-exchange protocols in which users authenticate themselves by proving possession of credentials satisfying arbitrary policies, instead of using the more traditional mechanism of a public-key infrastructure. Definitions in the universal composability framework are given, and practical protocols satisfying these definitions, for policies of practical interest, are presented. All protocols are analyzed in the common reference string model, assuming adaptive corruptions with erasures, and no random oracles. The new security notion includes password-authenticated key exchange as a special case, and new, practical protocols for this problem are presented as well, including the first such protocol that provides resilience against server compromise (without random oracles).

1 Introduction

Secure two-party authentication and key exchange are fundamental problems. Traditionally, the parties authenticate each other by means of their identities, using a public-key infrastructure (PKI). However, this is not always feasible or desirable: an appropriate PKI may not be available, or the parties may want to remain anonymous, and not reveal their identities.

To address these needs, we introduce the notion of credential-authenticated identification (CAID) and key exchange key exchange (CAKE), where the compatibility of the parties' *credentials* is the criteria for authentication, rather than the parties' *identities* relative to some PKI.

We assume that prior to the protocol, the parties agree upon a *policy*, which specifies the types of credentials they each should hold, along with additional constraints that each credential should satisfy, and (possibly) relationships that should hold *between* the two credentials. The protocol should then determine whether or not the two parties have credentials that satisfy the policy, and in the CAKE case, should generate a session key, which could then be used to implement a secure communication session between the two parties. In any case, neither party should learn anything else about the other party's credentials, other than whether or not they satisfied the policy.

For example, Alice and Bob may agree on a policy that says that Alice should hold an electronic ID card that says her age is at least 18, and that Bob should hold a valid electronic library card. If Alice then inputs an appropriate ID card

and Bob inputs an appropriate library card, the protocol should succeed, and, in the CAKE case, both parties should obtain a session key. However, if, say, Alice tries to run the protocol without an appropriate ID card, the protocol should fail; moreover, Alice should not learn anything at all about Bob's input; in particular, Alice should not even be able to tell whether Bob had a library card or not.

As mentioned above, we may even consider policies that require that certain relationships hold between the two credentials. For example, Alice and Bob may agree upon a policy that says that they both should have national ID cards, and that they should live in the same state.

Both of the two previous examples illustrate that the CAKE problem is closely related to the "secret handshake" problem. In the latter problem, two parties wish to determine if they belong to the same group, so that neither party's status as a group member is revealed to the other, unless both parties belong to the same group. There are many papers on secret handshakes (see [12] for a recent paper, and the references therein). The system setup assumptions and security requirements vary significantly among the papers in the secret handshakes literature, and so we do not attempt a formal comparison of CAKE with secret handshakes. Nevertheless, the two problems share a common motivation, and to the extent that one can view owning a credential as belonging to a group, the two problems are very similar.

We also observe that the CAKE problem essentially includes the PAKE (password-authenticated key exchange) problem as a special case: the credentials are just passwords, and the policy says that the two passwords must be equal.

Our Contributions. So that our results are as general as possible, we work in the Universal Composability (UC) framework of Canetti [7]. We give natural ideal functionalities for CAID and CAKE, and give efficient, modularly designed protocols that realize these functionalities. If the underlying credential system is practical and comes equipped with practical proof-of-ownership protocols (such as the IDEMIX system, based on Camenisch and Lysyanskaya [5]), and if the policies are not too complex, the resulting CAKE protocols are fairly practical. In addition, if the credential system provides extra features such as traceability or revocability, or other mechanisms that mitigate against unwanted "credential sharing", then our protocols inherit these features as well.

All of our protocols are proved UC-secure in the adaptive corruption model, assuming parties can effectively erase internal data. Our protocols require a common reference string (CRS), but otherwise make use of standard cryptographic assumptions, and do not rely on random oracles.

As mentioned above, CAKE includes PAKE, and we also obtain two new practical PAKE protocols. The first is a practical PAKE protocol that is secure in the adaptive corruption model (with erasures); this is not the first such protocol (this was achieved recently by Abdalla, Chevalier, and Pointcheval [1], using completely different techniques). The second PAKE protocol is a simple variant of the first, but provides security against server compromise: the protocol is an asymmetric protocol run between a client, who knows the password, and a server,

who only stores a function of the password; if the server is compromised, it is still hard to recover the password. Our new protocol is the first fairly practical PAKE protocol (UC-secure or otherwise) that is secure against server compromise and that does not rely on random oracles. Previous practical PAKE protocols that provide security against server compromise (such as Gentry, MacKenzie, and Ramzan [11]) all relied on random oracles (and also were analyzed only in the static corruption model).

Outline of the paper. In §2, we provide some background on the UC framework; in addition, we provide some recommendations for improving some of the low-level mechanics of the UC framework, to address some minor problems with the existing formulation in [7] that were uncovered in the course of this work. In any case, our results can be understood independently of these recommendations.

In §3, we introduce ideal functionalities for *strong* CAID and CAKE. These ideal functionalities are stronger than we want, as they can only be realized by protocols that use authenticated channels. Nevertheless, they serve as a useful building block. We also discuss there the types of policies that will be of interest to us here, as we want to restrict our attention to policies that are useful and that admit practical protocols.

In §4, we show how a protocol that realizes the strong CAID or CAKE functionalities can be easily and efficiently transformed into a protocol that realizes the CAID and CAKE functionality. The resulting protocol does not rely on authenticated channels. To this end, we utilize the idea of "split functionalities", introduced in [2]. Although the idea of using split functionalities for nonstandard authentication mechanisms was briefly mentioned in [2], it was not pursued there, and no new types of authentication protocols were presented. In this section, we review the basic notions introduced in [2], adjusting the definitions and results slightly to better meet our needs. We also give some new constructions, which are simpler and more efficient in the two-party setting.

In §5 we review definitions of UC zero knowledge (UCZK), and provide some new definitions that will be useful to us. UCZK will be a critical building block in the design of our CAID/CAKE protocols. In this section, we discuss a general language of statements we will want to be able to prove, as well as practical implementations of UCZK protocols for proving such statements. In a companion paper, we plan on fleshing out the details of this general framework, but it should be clear, based on these discussions, that there are, in fact, practical UCZK protocols for all the statements we need to prove in our CAID/CAKE protocols.

In §6, we present practical strong CAID/CAKE protocols for some fairly general policies of interest, and prove their security in the UC-framework, assuming secure channels. Using the split functionalities ideal in §4, these protocols can be transformed into practical CAID/CAKE protocols, which do not assume secure channels.

In §7, we present practical strong CAID/CAKE protocols for the equality relation and an interesting relation related to discrete logarithms. The former gives rise to our first new PAKE protocol, while the latter gives rise to our second new PAKE protocol (which provides resilience against server compromise).

Due to space limitations, many details, and all proofs, are left to the full paper [4].

2 Some UC background

Our corruption model is always *adaptive corruptions with erasures*. We believe that allowing adaptive corruptions is important — there are known examples of protocols that are secure with respect to static corruptions, but trivially insecure if adaptive corruptions are allowed. Allowing erasures is a bit of a compromise: on the one hand, properly implementing secure erasures is difficult — but not impossible; on the other hand, if erasures are not allowed, then it becomes very difficult to obtain truly practical protocols, leading to results that are of theoretical interest only.

To streamline the descriptions of ideal functionalities, we assume the following convention in any two-party ideal functionality: *the adversary may at any time tell the ideal functionality to abort the protocol for one of the parties — the ideal functionality sends the special message* `abort` *to that party, and does not communicate any further with that party.*

In an actual protocol, an `abort` output would be generated when a "time out" or "error" condition was detected; the aborting party will also erase all internal data, and all future incoming messages will be ignored. While not essential for modeling security, it does allow us to distinguish between detectable and undetectable unfairness in protocols.

We clarify here a number of issues regarding terminology and notation in the UC framework. By a ***party*** we always mean an *interactive Turing machine (ITM)*. A party P is addressed by ***party ID (PID)*** and ***session ID (SID)***. So if P has PID P_{pid} and SID P_{sid}, then the PID/SID pair $(P_{\text{pid}}, P_{\text{sid}})$ uniquely identifies the party: no two parties in the system may have the same PID/SID pair. The convention is that the participants of any single protocol instance share the same SID, and conversely, if two parties share the same SID, then they are regarded as participants in the same protocol.

In [7] there are no semantics associated with PIDs, other than their role to distinguish participants in a protocol instance. Some authors (sometimes implicitly) tend to use the term "party" to refer to all ITMs that share a PID. We shall not do this: **a party is just a single ITM** (but see §2.2 below).

In describing protocols and ideal functionalities, we generally omit SIDs in messages — these can always be assumed to be implicitly defined.

2.1 Notions of security. We recall some basic security notions from [7], with some extensions in [10] and [2]. We will not be too formal here.

We say that a protocol Π ***realizes*** a protocol Π^*, if for every adversary A, there exists an adversary (i.e., simulator) A^*, such that for every environment Z, Z cannot distinguish an attack of A on Π from an attack of A^* on Π^*. Here, Z is allowed to interact directly with the adversary and (via subroutine input/output) with parties (running the code for Π or Π^*) that share the same SID.

If we like, we can remove the restriction that parties must share the same SID, which effectively allows Z to interact with multiple, concurrently running instances of a single protocol in the above experiment. With this relaxation, we say that Π ***multi-realizes*** Π^*. If these multiple instances of Π access a *common* instance of a setup functionality $\mathcal{G}$, then we say that Π multi-realizes Π^* ***with joint access to*** $\mathcal{G}$. In applications, $\mathcal{G}$ is typically a common reference string (CRS).

The UC Theorem [7] implies that if Π realizes Π^*, then Π multi-realizes Π^*. However, if Π makes use of a setup functionality $\mathcal{G}$, then it does not necessarily follow that Π multi-realizes Π^* with joint access to $\mathcal{G}$: one typically has to analyze the multiple-instance experiment directly.

In the above definitions, if Π^* is the ideal protocol associated with an ideal functionality $\mathcal{F}$, then we simply say that Π (multi-)realizes $\mathcal{F}$ (with joint access to $\mathcal{G}$). We also have some simple transitivity properties: if Π_1 realizes Π_2, and Π_2 realizes Π_3, then Π_1 realizes Π_3; also, if Π_1 multi-realizes Π_2 with joint access to $\mathcal{G}$, and Π_2 realizes Π_3, then Π_1 multi-realizes Π_3 with joint access to $\mathcal{G}$.

A protocol Π may itself make use of an ideal functionality $\mathcal{F}'$ as a subroutine (where an instance of Π may make use of multiple, independent instances of $\mathcal{F}'$). In this case, we call Π an ***$\mathcal{F}'$-hybrid protocol***. We may modify Π by instantiating each instance of $\mathcal{F}'$ with an instance of a protocol Π', and we denote the modified version of Π by $\Pi[\mathcal{F}'/\Pi']$. The UC Theorem implies that if Π' realizes $\mathcal{F}'$, then $\Pi[\mathcal{F}'/\Pi']$ realizes Π. Also, if Π' multi-realizes $\mathcal{F}'$ with joint access to $\mathcal{G}$, then $\Pi[\mathcal{F}'/\Pi']$ multi-realizes Π with joint access to $\mathcal{G}$.

This last statement is essentially a reformulation of a special case of the JUC Theorem [10], but in a form that is more convenient to apply. The notion of multi-realization (introduced, somewhat informally, in [2], and which can be easily expressed in the Generalized UC (GUC) framework [8]) seems a more elegant and direct way of modeling joint access to a CRS or similar setup functionality.

2.2 Conventions regarding SIDs. We shall assume that an SID is structured as a ***pathname***: $name_0/name_1/\cdots/name_k$. These pathnames reflect the subroutine call stack: when an honest party invokes an instance of subprotocol as a separate party, the new party has the same PID of the invoking party, and the SID is extended on the right by one element. Furthermore, we shall assume for two-party protocols, the rightmost element $name_k$, called the ***basename***, has the form $ext : P_{\text{pid}} : Q_{\text{pid}} : data$, where ext is a "local name" used to ensure unique basenames, P_{pid} and Q_{pid} are the PIDs of the participants P and Q, and $data$ represents shared public parameters. The ordering of these PIDs can be important in protocols where the two participants play different roles.

These conventions streamline and clarify a number of things. In application of the UC Theorem, we will be interested exclusively in protocols that act as subroutines: they are explicitly invoked by a single caller, who provides all inputs, and who receives all outputs.

The main points here are: (i) a subroutine is explicitly invoked by the caller, and (ii) the callee implicitly knows where to write its output. We can (and will)

design protocols that deviate from this simple subroutine structure, although the UC Theorem will not directly apply in these cases.

With these restrictions, it also is convenient to make some restrictions on ideal functionalities: *we shall assume that an ideal functionality only delivers an output to a party that has previously supplied the ideal functionality with an input.*

These conventions are simply self-imposed restrictions, and do not represent a modification of the UC framework itself. However, in the full paper, we discuss some modifications to the UC framework that strictly impose these restrictions, along with a few other rules. Our rules guarantee that if P is a party with PID *pid* and SID *sid*, and if P' is a party with PID *pid* and SID *sid*/*basename*, then P' is a subroutine of P that was created by P, and moreover, so long as P remains honest, then so does P'. As discussed in the full paper, we believe that without some type of restrictions such as these, there are some fundamental problems with the UC framework itself.

2.3 System parameters. A common reference string, or CRS, is sometimes very useful. Sometimes, however, a different, but related notion is useful: a ***system parameter***. Like a CRS, a system parameter is assumed to be generated by a trusted party, but unlike a system parameter, a CRS is visible to *all* parties, including the environment. A nice way to model this is using some elements of the GUC framework (although we do not attempt to design any protcols that achieve full GUC security here).

In designing a protocol that realizes some ideal functionality, a system parameter is a much better type of setup functionality than a CRS, as the security properties of protocols that use a CRS are not always so clear (e.g., "deniability" — see discussion in [8]). These problems do not arise with system parameters. Moreover, if a protocol Π realizes an ideal functionality $\mathcal{F}$ using a system parameter, then it is easy to see that Π multi-realizes $\mathcal{F}$ as well — there is no need to separately analyze a multi-instance experiment. A system parameter can also be used to parameterize an ideal functionality — a CRS cannot be used for this purpose, as that would conflate specification and implementation.

We can distinguish between two types of system parameters: ***public coin*** and ***private coin***. In a public-coin system parameter, even the random bits used to generate the system parameter are visible to the environment (but no one else). In a private-coin system parameter, the random bits used to generate the system parameter remain hidden from all parties.

2.4 Authenticated channels. We present here an ideal functionality for an authenticated channel. We have tuned this functionality to adhere to our conventions. We call this ***ideal functionality*** $\mathcal{F}_{\mathbf{ach}}$.

For an SID is of the form $sid := parent/ext : P_{\text{pid}} : Q_{\text{pid}} :$, where P is the sender and Q is the receiver, and for an adversary A, the ideal functionality $\mathcal{F}_{\text{ach}}$ runs as follows:

1. *Wait for both:* (a) *an input message* $(\texttt{send}, x)$ *from* P*, then send* $(\texttt{send}, x)$ *to* A*;* (b) *an input message* `ready` *from* Q*, then send* `ready` *to* A.

2. *Wait for the message* `deliver` *from A, then send the output message* $(\texttt{deliver}, x)$ *to Q.*

Corruption rule: *If P is corrupted between Steps 1a and 2, then A is allowed to change the value of x (at any time before Step 2).*

NOTES: *(i)* Like the corresponding functionality in [7], this one allows delivery of a single message per session. Multiple sessions should be used to send multiple messages. Alternatively, one could also define a multi-message functionality. *(ii)* Unlike the corresponding functionality in [7], the receiver here must explicitly initialize the channel before receiving a message. This design conforms to our conventions stated above, and is further discussed in the full paper.

2.5 Secure channels. Secure channels provide both authentication and secrecy. We present a ideal functionality that is tuned to adhere to our conventions, and to our adaptive corruptions with erasures assumption.

One way to define secure channels is to modify $\mathcal{F}_{\text{ach}}$ as follows: in Step 1, send $(\texttt{send}, \text{len}(x))$ to A, and in the corruption rule, A is given x and allowed to modify it x as well. Here, $\text{len}(x)$ is the length of x. However, it turns out that a different functionality can be implemented more efficiently:

1. *Wait for both:* (a) *an input message* $(\texttt{send}, x)$ *from P, then send the message* $(\texttt{send}, \mathit{len}(x))$ *to A;* (b) *an input message* $(\texttt{ready}, \mathit{maxlen})$ *from Q, then send the message* $(\texttt{ready}, \mathit{maxlen})$ *to A.*
2. *Wait for the message* `lock` *from A; verify that* $\mathit{len}(x) \leq \mathit{maxlen}$*; if not, halt.*
3. *Wait for both:* (a) *a message* `done` *from A, then send the output message* `done` *to P;* (b) *a message* `deliver` *from A, then send the output message* $(\texttt{deliver}, x)$ *to Q.*

Corruption rule: *If P is corrupted between Steps 1a and 2, then A is given x and is allowed to change the value of x (at any time before Step 2).*

We call this ***ideal functionality*** $\boldsymbol{\mathcal{F}_{\text{sch}}}$**.** Here, the receiver specifies the maximum length message he is prepared to accept. This functionality reflects the fact that most of the time, the receiver knows the general "size and shape" of the message it is expecting, and so no additional interaction is required. In the cases where this information is not known in advance, the sender can transmit the length information to the receiver ahead of time on an authenticated channel.

3 Ideal functionalities for strong CAID and CAKE

In this section, we present ideal functionalities for *strong* CAID and CAKE. These ideal functionalities are stronger than we want, as they can only be realized by protocols that use authenticated channels. However, in the next section, we discuss how to we can very easily modify such protocols to obtain protocols that realize the desired CAID/CAKE functionalities (which will be defined in terms of strong CAID/CAKE).

We start with strong CAID. At a high level, the ideal functionality for strong CAID, denoted $\mathcal{F}^*_{\text{caid}}$, works as follows. We have two parties, P and Q. P and

Q agree (somehow) on a binary relation R, which consists of a set of pairs (s, t). Then P and Q submit values to the ideal functionality: P submits a value s and Q a value t. The ideal functionality then checks if $(s, t) \in R$; if so, it sends P and Q the value 1, and otherwise the value 0. The relation R represents the "policy", discussed in §1.

The above description is lacking in details: some essential, and others not. We now describe some detailed variants of the above general idea. We assume that party P has PID P_{pid} and SID P_{sid}. Likewise, we assume that party Q has PID Q_{pid} and SID Q_{sid}.

3.1 Ideal Functionality $\mathcal{F}^*_{\text{caid}}$. We assume that the SIDs of the two parties are of the form $\mathit{parent}/\mathit{ext} : P_{\text{pid}} : Q_{\text{pid}} : \langle\, R \,\rangle$, where $\langle\, R \,\rangle$ is a description of the relation R. In principle, any efficiently computable family of relations is allowable, but specific realizations may implement only relations from some specific family of relations. It will convenient to assume that the special symbol $\perp$ has the following semantics: for all s, t, neither $(\perp, t)$ nor $(s, \perp)$ are in R.

An instance of $\mathcal{F}^*_{\text{caid}}$ with SID $\mathit{parent}/\mathit{ext} : P_{\text{pid}} : Q_{\text{pid}} : \langle\, R \,\rangle$ runs as follows.

1. *Wait for both:* (a) *an input message* $(\texttt{left-input}, s)$ *from* P*, then send* `left-input` *to* A*;* (b) *an input message* $(\texttt{right-input}, t)$ *from* Q*, then send* `right-input` *to* A*.*
2. *Wait for a message* `lock` *from* A*; set* res *to* 1 *if* $(s, t) \in R$*, and* 0 *otherwise.*
3. *Wait for both:* (a) *a message* `deliver-left` *from* A*, then send the output message* $(\texttt{return}, \mathit{res})$ *to* P*;* (b) *a message* `deliver-right` *from* A*, then send the output message* $(\texttt{return}, \mathit{res})$ *to* Q*.*

Corruption rules: (i) *If* P *(resp.,* Q*) is corrupted between Steps 1a (resp., 1b) and 2, then* A *is given* s *(resp.,* t*), and is allowed to change the value of* s *(resp.,* t*) at any time before Step 2.* (ii) *If* P *(resp.,* Q*) is corrupted between Steps 2 and Steps 3a (resp., 3b), then* A *is given* s *(resp.,* t*).*

Note that the inclusion of $P_{\text{pid}}, Q_{\text{pid}}$ in the SID serves to break symmetry, and establish P as the "left" party and Q as the "right" party. The above ideal functionality captures the inherent "unfairness" in any such protocol: if one party is corrupt, they may learn that the relation holds, while the other may not. However, such unfairness is at least detectable: since we do not conflate `abort` with a result of 0, if any party is being treated unfairly, this will at least be detected by an `abort` message. One could consider a weaker notion of security, in which 0 and `abort` were represented by the same value. While this may allow for more efficient protocols, such protocols may allow "undetectable unfairness". With our present formulation, a result of `abort` may indicate an unfair run of the protocol (or it may just indicate that there are network problems). The functionality $\mathcal{F}^*_{\text{caid}}$ does not provide as much privacy as one might like; in particular, if P and Q are honest, then A still learns the relation R. In the full version of the paper, we discuss variations that prevent this.

3.2 From authentication to key exchange. Functionality $\mathcal{F}^*_{\text{caid}}$ may be extended to provide key exchange in addition to authentication modifying Step 2 as follows:

2. *Wait for a message* $(\texttt{lock}, K_{\mathrm{adv}})$ *from A; then set res to* $(1, K)$ *if* $(s,t) \in R$*, and* 0 *otherwise, where the key K is determined as follows: if either P or Q are currently corrupted, set* $K := K_{\mathrm{adv}}$*; otherwise, generate K at random (according to some prescribed distribution).*

Corruption rules are unchanged. We call this ***ideal functionality*** $\boldsymbol{\mathcal{F}^{*}_{\mathbf{cake}}}$.

3.3 Some relations of interest. One type of relation that is of particular interest is a simple ***product relation***, where $R = S \times T$. For example, we may have $S = \{s : (x,s) \in E\}$, for a given x and a fixed relation E. Here, s might be an "anonymous credential" issued by some authority whose public key is x; the relation E would assert that s is a valid credential relative to x, possibly satisfying some other constraints as well.

A well-known example of an anonymous credential system of this type is the IDEMIX system [5]. This system comes with efficient zero-knowledge protocols for proofs of possession of credentials that we will be able to exploit. IDEMIX may also be equipped with mechanisms for identity escrow, revocation, etc., which automatically enhances the functionality of any strong CAID/CAKE protocol.

Similarly, we may have $T = \{t : (y,t) \in F\}$, for a given y and fixed relation F. In this case the description $\langle\, R \,\rangle$ of R is the pair (x,y).

Two generalizations of potential interest are as follows. First, suppose we have binary relations $R_1, \ldots, R_k$. We can define their ***vectored union*** as the binary relation $R = \{((s_1, \ldots, s_k), (t_1, \ldots, t_k)) : (s_i, t_i) \in R_i \text{ for some } i = 1\,..\,k\}$. For example, each relation R_i may represent a pair of "compatible" credentials, and the protocol should succeed if the two parties hold one such pair between them. Or more simply, the two parties may agree on a list of "clubs", and then determine if there is any one club to which they both belong.

Second, we might consider the intersection of a product relation with a ***partial equality relation***: $\{(s,t) : \sigma(s) = \tau(t)\}$, where σ and τ are appropriate functions. Such relations can usefully model the "secret handshake" scenario, where $\sigma(s)$ and $\tau(t)$ perhaps represent "group names". A special case of this, of course, is the equality relation. A CAKE protocol for equality is essentially a PAKE protocol — this is discussed in §7.

One might even combine the above, considering vectored unions of such intersections. The reason for singling out these types of relations is that they are of potential practical interest, and admit efficient protocols.

4 Bootstrapping an authentication protocol

We shall presently give efficient protocols that realize strong CAID/CAKE functionalities for various relations of interest. All of these protocols work assuming secure channels. Of course, this is not interesting by itself, since we really want to use these protocols to establish secure channels in a setting without any existing authentication mechanism.

Without at least authenticated channels, it is impossible to realize strong CAID/CAKE. The solution is to weaken the notion of security, using the idea

of "split functionalities", introduced in [2]. Our definitions of the CAID/CAKE functionalities are simply the split versions of the strong CAID/CAKE functionalities.

Although the idea of using split functionalities for nonstandard authentication mechanisms was briefly mentioned in [2], it was not pursued there, and no new types of authentication protocols were presented. In this section, we review the basic notions introduced in [2], adjusting the definitions and results slightly to better meet our needs, and give some new constructions, as well.

4.1 Details: split functionalities. We give a slight reformulation of the definitions and results in [2]: we focus on the two-party case, and we also make a few small syntactic changes that will allow us to apply the results in a more convenient way.

The basic idea is the same as in [2]. If $\mathcal{F}$ is a two-party ideal functionality involving two parties, P and Q, then the split functionality $\mathsf{s}\mathcal{F}$ works roughly as follows. Before any computation begins, the adversary partitions the set $\{P, Q\}$ into ***authentication sets***: in the two-party case, the authentication sets are either $\{P\}$ and $\{Q\}$, or the single authentication set $\{P, Q\}$. The parties within an authentication set access a common instance of $\mathcal{F}$, while parties in different authentication sets access independent instances of $\mathcal{F}$. This is achieved by "mangling" SIDs appropriately: each authentication set is assigned a unique "channel ID" *chid*, which is used to "mangle" the SIDs of the instances of $\mathcal{F}$. Thus, the most damage an adversary can do is to make P and Q run two *independent* instances of $\mathcal{F}$.

As we shall see, one can transform any protocol Π that realizes $\mathcal{F}$, where Π relies on authenticated and/or secure channels, into a protocol $\mathsf{s}\Pi$ that realizes $\mathsf{s}\mathcal{F}$, where $\mathsf{s}\Pi$ relies on neither authenticated nor secure channels. Moreover, $\mathsf{s}\Pi$ is almost as efficient as Π. This result was first proved in [2]; however, we give a more efficient transformation — based on Diffie-Hellman key exchange — that is better suited to the two-party case.

Our CAID/CAKE functionalities are simply defined as the split versions of the strong CAID/CAKE functionalities: $\mathsf{s}\mathcal{F}^*_{\text{caid}}$ and $\mathsf{s}\mathcal{F}^*_{\text{cake}}$. Protocols for these functionalities may be obtained by applying the split transformation to the protocols for the corresponding strong functionalities.

4.2 General split functionalities. Now we give the general split functionality in more detail. Let $\mathcal{F}$ be an ideal functionality for a two party protocol. As in §2.2, we assume that the SID for $\mathcal{F}$ is of the form $\mathit{parent}/\mathit{ext} : P_{\text{pid}} : Q_{\text{pid}} : \mathit{data}$, and that $\mathcal{F}$ never generates an output for a party before receiving an input from that party.

The ***split functionality*** $\mathsf{s}\mathcal{F}$ has an SID $s := \mathit{parent}/\mathit{ext} : P_{\text{pid}} : Q_{\text{pid}} : \mathit{data}$ of the same form as $\mathcal{F}$, and for an adversary A runs as follows.

- *Upon receiving a message* `init` *from a party* $X \in \{P, Q\}$*: record* $(\texttt{init}, X_{\text{pid}})$*, send* $(\texttt{init}, X_{\text{pid}})$ *to* A*.*
- *Upon receiving a message* $(\texttt{authorize}, X_{\text{pid}}, \mathcal{H}, \mathit{chid})$ *from* A*, such that*

(1) X_{pid} *is the PID of some* $X \in \{P, Q\}$; (2) $\{X_{\text{pid}}\} \subseteq \mathcal{H} \subseteq \{P_{\text{pid}}, Q_{\text{pid}}\}$; (3) $(\texttt{init}, X_{\text{pid}})$ *has been recorded;* (4) *no tuple* $(\texttt{authorize}, X_{\text{pid}}, \ldots)$ *has been recorded; and* (5) *if a tuple* $(\texttt{authorize}, X'_{\text{pid}}, \mathcal{H}', \mathit{chid}')$ *has been recorded, then either (a)* $\mathcal{H}' = \mathcal{H}$ *and* $\mathit{chid}' = \mathit{chid}$ *or (b)* $\mathcal{H}' \cap \mathcal{H} = \emptyset$ *and* $\mathit{chid}' \neq \mathit{chid}$

do the following:

(1) *if no tuple of the form* $(\texttt{authorize}, \cdot, \mathcal{H}, \mathit{chid})$ *has already been recorded, then initialize a "virtual" instance of* $\mathcal{F}$ *with SID* $\mathit{sid}_{\mathcal{H}} := \mathit{chid}/\mathit{sid}$*; we denote this instance* $\mathcal{F}_{\mathcal{H}}$ *and define* $\mathit{chid}_{\mathcal{H}} := \mathit{chid}$*; in addition, for each* $Y \in \{P, Q\}$*, if* $Y_{\text{pid}} \notin \mathcal{H}$ *or* Y *is corrupt, then notify* $\mathcal{F}_{\mathcal{H}}$ *that the party with PID* Y_{pid} *and SID* $\mathit{sid}_{\mathcal{H}}$ *is corrupt, and forward to* A *the response of* $\mathcal{F}_{\mathcal{H}}$ *to this notification;* (2) *record the tuple* $(\texttt{authorize}, X_{\text{pid}}, \mathcal{H}, \mathit{chid})$; (3) *send the output message* $(\texttt{authorize}, \mathit{chid})$ *to* X.

- *Upon receiving a message* $(\texttt{input}, v)$ *from* $X \in \{P, Q\}$*, such that a tuple* $(\texttt{authorize}, X_{\text{pid}}, \mathcal{H}, \mathit{chid})$ *has been recorded: send the message* v *to* $\mathcal{F}_{\mathcal{H}}$*, as if coming as an input from the party with PID* X_{pid} *and SID* $\mathit{sid}_{\mathcal{H}}$.
- *Upon receiving a message* $(\texttt{input}, X_{\text{pid}}, \mathcal{H}, v)$ *from* A*, such that*

  (1) X_{pid} *is the PID of some* $X \in \{P, Q\}$, (2) *a (uniquely determined) instance* $\mathcal{F}_{\mathcal{H}}$ *with* $X_{\text{pid}} \in \mathcal{H}$ *has been initialized; and,* (3) $X_{\text{pid}} \notin \mathcal{H}$

  send the message v *to* $\mathcal{F}_{\mathcal{H}}$*, as if coming as an input from the party with PID* X_{pid} *and SID* $\mathit{sid}_{\mathcal{H}}$.
- *Whenever an instance* $\mathcal{F}_{\mathcal{H}}$ *delivers an output* v *to a party with PID* X_{pid}*, where* X_{pid} *is the PID of some* $X \in \{P, Q\}$*, do the following: if* $X_{\text{pid}} \in \mathcal{H}$*, then send the output message* $(\texttt{output}, v)$ *to* X*, else send the output message* $(\texttt{output}, X_{\text{pid}}, v)$ *to* A.
- *Upon receiving notification that a party* $X \in \{P, Q\}$ *is corrupted, such that a (uniquely determined) instance* $\mathcal{F}_{\mathcal{H}}$ *with* $X_{\text{pid}} \in \mathcal{H}$ *has been initialized: notify* $\mathcal{F}_{\mathcal{H}}$ *that the party with PID* X_{pid} *and SID* $\mathit{sid}_{\mathcal{H}}$ *is corrupted, and forward to* A *the response of* $\mathcal{F}_{\mathcal{H}}$ *to this notification.*

We have a slightly different formulation of split functionalities than in [2], but the differences are mainly syntactic — our method of mangling the SIDs fits nicely in to our set of conventions on SIDs. In addition, in [2], a party is allowed to send an input as long as its authentication set is defined, whereas we require that a party wait for its explicit authorization notification before proceeding. This seems to avoid some potential confusion.

4.3 A multi-session secure channels functionality. We need a "multi-session extension" of our ideal functionality for secure channels. One approach would be to use the definition in [10]. However, a direct application of that definition would be unworkable, for two reasons: first, it would require that any implementation keep track of all subsession IDs that were ever used; second, the multi-session extension applies to all possible parties, whereas, we can really only deal with the same two parties in all subsessions. So for these reasons, we present our own multi-session extension, which we denote $\mathcal{F}_{\text{msc}}$. Note that in addition to secure channels (corresponding to the functionality $\mathcal{F}_{\text{sch}}$), it also provides for channels that only provide authentication (corresponding to the functionality $\mathcal{F}_{\text{ach}}$). It is quite tedious, and not very enlightening. The details are in the full paper.

4.4 Split key exchange. We now discuss a simple, low-level primitive: split key exchange. Let $\mathcal{K}$ be a key set. The ***ideal functionality $\mathcal{F}_{\mathbf{ske}}$*** (parameterized by $\mathcal{K}$) has an SID of the form $parent/ext : P_{\text{pid}} : Q_{\text{pid}} :$, and for an adversary A, runs as follows:

- *Upon receiving a message* $\texttt{init}$ *from a party* $X \in \{P, Q\}$*: record* $(\texttt{init}, X_{\text{pid}})$*, send* $(\texttt{init}, X_{\text{pid}})$ *to* A*.*
- *Upon receiving a message* $(\texttt{authorize}, X_{\text{pid}}, \mathcal{H}, chid, K)$ *from* A*, such that*
  (1) X_{pid} *is the PID of some* $X \in \{P, Q\}$*;* (2) $\{X_{\text{pid}}\} \subseteq \mathcal{H} \subseteq \{P_{\text{pid}}, Q_{\text{pid}}\}$*;* (3) $K \in \mathcal{K}$*;* (4) $(\texttt{init}, X_{\text{pid}})$ *has been recorded;* (5) *no tuple* $(\texttt{authorize}, X_{\text{pid}}, \ldots)$ *has been recorded; and,* (6) *if a tuple* $(\texttt{authorize}, X'_{\text{pid}}, \mathcal{H}', chid', K')$ *has been recorded, then either (a)* $\mathcal{H}' = \mathcal{H}$ *and* $chid' = chid$ *or (b)* $\mathcal{H}' \cap \mathcal{H} = \emptyset$ *and* $chid' \neq chid$

  do the following:
  (1) *record the tuple* $(\texttt{authorize}, X_{\text{pid}}, \mathcal{H}, chid, K)$*;* (2) *if* $K_{\mathcal{H}}$ *is not yet defined, then define it as follows: if* $\mathcal{H} = \{X_{\text{pid}}\}$*, then* $K_{\mathcal{H}} \leftarrow K$*, else* $K_{\mathcal{H}} \leftarrow_{\text{R}} \mathcal{K}$*;* (3) *send the output message* $(\texttt{key}, chid, K_{\mathcal{H}})$ *to* X*.*

We now present a simple protocol, Π_{ske}, that realizes the functionality $\mathcal{F}_{\text{ske}}$, under the decisional Diffie-Hellman (DDH) assumption. Assume a group $\mathbb{G}$ of prime order q generated by $g \in \mathbb{G}$ where the DDH holds. The description of $\mathbb{G}$, q, and g is viewed here as a system parameter. We also assume a PRG that maps a random $w \in \mathbb{G}$ to a pair of keys $(K, K_{\text{auth}}) \in \mathcal{K} \times \mathcal{K}_{\text{auth}}$, where $\mathcal{K}_{\text{auth}}$ is some large set.

For two parties P and Q with SID $sid := parent/ext : P_{\text{pid}} : Q_{\text{pid}} :$, ***protocol $\boldsymbol{\Pi}_{\mathbf{ske}}$*** runs as follows. The roles played by P and Q are asymmetric. The protocol for P runs as follows:

1. *P waits for an input* $\texttt{init}$*; then it computes* $x \leftarrow_{\text{R}} \mathbb{Z}_q$*,* $u \leftarrow g^x$*, and sends* u *to* Q*.*
2. *P waits for* $v \in \mathbb{G}$ *from* Q*; then it computes* $w \leftarrow v^x$*, derives keys* K, K_{auth} *from* w *using the PRG, sets* $chid \leftarrow \langle u, v \rangle$*, sends the key* K_{auth} *to* Q *(after erasing all internal state other than* $chid$ *and* K*).*
3. *P waits for a continuation signal, and then outputs and outputs* $(\texttt{key}, chid, K)$ *(after erasing all internal state).*

Note that in the UC framework, a party is allowed to only send one message at a time; therefore, P first sends a message to Q (via the adversary, of course), and then waits for a continuation signal (provided by the adversary) before delivering its own output.

The protocol for Q runs as follows:

1. *Q waits for an input* $\texttt{init}$*; then it then does nothing, except to notify the network (i.e., adversary) that it is ready.*
2. *Q waits for* $u \in \mathbb{G}$ *from* P*; then it computes* $y \leftarrow_{\text{R}} \mathbb{Z}_q$*,* $v \leftarrow g^y$*,* $w \leftarrow u^y$*, derives keys* K, K_{auth} *from* w*, erases* y, w*, sets* $chid \leftarrow \langle u, v \rangle$*, and sends* v *to* P*.*
3. *Q waits for* $K'_{\text{auth}} \in \mathcal{K}_{\text{auth}}$ *from* P*; then it tests if* $K_{\text{auth}} = K'_{\text{auth}}$*; if so, it outputs* $(\texttt{key}, chid, K)$ *(after erasing all internal state).*

Theorem 1. *Assuming the DDH for* $\mathbb{G}$*, an appropriate PRG, and assuming the set* $\mathcal{K}_{\text{auth}}$ *is large, protocol* Π_{ske} *realizes the ideal functionality* $\mathcal{F}_{\text{ske}}$*.*

4.5 Realizing split multi-session secure channels. Our goal now is to realize the split version $\mathsf{s}\mathcal{F}_{\text{msc}}$ of the multi-session secure channels functionality $\mathcal{F}_{\text{msc}}$ presented in §4.3. This will be done with an $\mathcal{F}_{\text{ske}}$-hybrid protocol Π_{smsc}, where $\mathcal{F}_{\text{ske}}$ is the split key exchange functionality discussed in §4.4. At a high-level, ***protocol*** $\boldsymbol{\Pi_{\text{smsc}}}$ works as follows:

1. *Wait for an input message* `init`*, then send the message* `init` *to* $\mathcal{F}_{\text{ske}}$.
2. *Wait for a message* $(\texttt{key}, \mathit{chid}, K)$ *from* $\mathcal{F}_{\text{ske}}$*; then do the following:*
   (a) *derive subkeys required to implement bidirectional secure channels, erasing the key* K*; these channels will be implemented using a variant of Beaver and Haber's technique [3] (see full paper).* (b) *generate the output message* $(\texttt{authorize}, \mathit{chid})$.
3. *Now use the keys derived in the previous step to process the secure channels logic.*

Theorem 2. *The* $\mathcal{F}_{\text{ske}}$*-hybrid protocol* Π_{smsc} *realizes the ideal functionality* $\mathsf{s}\mathcal{F}_{\text{msc}}$*, assuming a secure PRG and secure MAC.*

4.6 Realizing general split functionalities. Let $\mathcal{F}$ be an arbitrary two-party ideal functionality. Let $\mathcal{G}$ be a setup functionality, such as a CRS. Let Π be an $(\mathcal{F}_{\text{ach}}, \mathcal{F}_{\text{sch}})$-hybrid protocol that multi-realizes $\mathcal{F}$ with joint access to $\mathcal{G}$ (where $\mathcal{F}_{\text{ach}}$ is defined in §2.4 and $\mathcal{F}_{\text{sch}}$ is defined in §2.5).

Our goal is to use Π to design an $\mathsf{s}\mathcal{F}_{\text{msc}}$-hybrid protocol $\mathsf{s}\Pi$ that multi-realizes $\mathsf{s}\mathcal{F}$ with joint access to $\mathcal{G}$. The point is, $\mathsf{s}\Pi$ does not require secure channels. Moreover, instantiating $\mathsf{s}\mathcal{F}_{\text{msc}}$ with Π_{smsc}, we obtain the a protocol $\mathsf{s}\Pi[\mathsf{s}\mathcal{F}_{\text{msc}}/\Pi_{\text{smsc}}]$ that multi-realizes $\mathsf{s}\mathcal{F}$ with joint access to $\mathcal{G}$.

At a high level, ***protocol*** $\boldsymbol{\mathsf{s}\Pi}$ works as follows:

1. *Wait for an input message* `init`*, then send the message* `init` *to* $\mathsf{s}\mathcal{F}_{\text{msc}}$.
2. *Wait for a message* $(\texttt{authorze}, \mathit{chid})$ *from* $\mathsf{s}\mathcal{F}_{\text{msc}}$*; then do the following:* (a) *initialize a "virtual" instance of* Π*, assigning it a PID and SID that are the same as that of this protocol instance, except that the SID pathname is prefixed* chid*;* (b) *generate the output message* $(\texttt{authorize}, \mathit{chid})$.
3. *Proceed as follows:* (a) *process input requests by passing them to the virtual instance of* Π*;* (b) *pass along outputs of the virtual instance of* Π *as outputs of this protocol instance;* (c) *use* $\mathsf{s}\mathcal{F}_{\text{msc}}$ *to implement the secure channels used by the virtual instance of* Π.

Theorem 3. *If* Π *is an* $(\mathcal{F}_{\text{ach}}, \mathcal{F}_{\text{sch}})$*-hybrid protocol that multi-realizes* $\mathcal{F}$ *with joint access to* $\mathcal{G}$*, then* $\mathsf{s}\Pi$ *is an* $\mathsf{s}\mathcal{F}_{\text{msc}}$*-hybrid protocol that multi-realizes* $\mathsf{s}\mathcal{F}$ *with joint access to* $\mathcal{G}$.

This is essentially the same as the main technical result (Lemma 4.1) of [2], but there are some technical differences — see full paper for more discussion.

5 Practical UC zero knowledge

Before getting into strong CAID/CAKE protocols, we need to discuss an essential building block: practical protocols for UC ZK (zero knowledge). We will need a slightly stronger version of ZK, which we call "enhanced ZK". In the adaptive

corruptions with erasures model, this is no more difficult to realize than ordinary ZK.

Let R be a binary relation, consisting of pairs (x, w): for such a pair, x is called the "statement" and w is called the "witness".

Let $\ell : \{0,1\}^* \to \{0,1\}^*$ be an "information leakage" function. The SID for an enhanced ZK protocol is of the form $parent/ext : P_{\text{pid}} : Q_{\text{pid}} :$, where P is the prover and Q the verifier. For an adversary A, an instance of ***ideal functionality*** $\mathcal{F}_{\mathtt{ezk}}$ with SID $sid := parent/ext : P_{\text{pid}} : Q_{\text{pid}} :$ runs as follows:

1. *Wait for both:* (a) *an input message* $(\mathtt{send}, x, w)$ *from* P *such that* $(x, w) \in R$*, then send the message* $(\mathtt{send}, \ell(x))$ *to* A*;* (b) *an input message* `ready` *from* Q*, then send* `ready` *to* A*.*
2. *Wait for the message* `lock` *from* A*.*
3. *Wait for both:* (a) *a message* `done` *from* A*, then send the output message* `done` *to* P*;* (b) *a message* `deliver` *from* A*, then send the output message* $(\mathtt{deliver}, x)$ *to* Q*.*

Corruption rule: *If* P *is corrupted between Steps 1a and 2, then* A *is given* (x, w) *and is allowed to change the value of* (x, w) *to any value* $(x', w') \in R$ *(at any time before Step 2).*

Note the similarity with our secure channels functionality. Here, the functionality is parameterized by the information leakage function ℓ, which is used to model the fact that some information about x may be leaked to an eavesdropping adversary. Typically, this information will be some rough information about the "size and shape" of x that ultimately determines the lengths of the ciphertexts that must be sent in an implementation.

Parameterized relations. In the above discussion, the relation R was considered to be a fixed relation. However, for many applications, it is convenient to let R be parameterized by a some system parameter (see §2.3). To realize the ZK (or extended ZK) functionality, it may be necessary to assume that the system parameter was generated in a certain way.

For example, a ZK protocol might require that the system parameter contains an RSA modulus N that is the product of two primes. To realize the ZK ideal functionality, it might not even be necessary that the factorization of N remain hidden. In such a case, the system parameter might be profitably viewed as a public-coin system parameter. This means that the environment may know the factorization of N, which may be useful to model situations where the factorization of N is used, say, to sign messages in higher-level protocols that use a ZK protocol as a subprotocol.

5.1 Practical protocols. Practical ZK protocols exist for the types of relations that we will be needed in our strong CAID/CAKE protocols — indeed, our protocols were designed with such protocols specifically in mind. In a companion paper, we give a detailed account of the current state of the art for such protocols. Here, we give a very brief sketch — see the full paper for more details.

We will be proving statements of the form

$$\nexists\, w_1 \in \mathcal{D}_1, \ldots, w_n \in \mathcal{D}_n : \phi(w_1, \ldots, w_n). \tag{1}$$

Here, we use the symbol "$\nexists$" instead of "$\exists$" to indicate that we are proving "knowledge" of a witness, rather than just its existence. The $\mathcal{D}_i$'s are domains which are finite intervals of integers centered around 0. ϕ is a predicate — we will presently place restrictions on the form of the domains and the predicate. A witness for a statement of the form (1) is a tuple $(w_1, \ldots, w_n)$ of integers such that $w_i \in \mathcal{D}_i$ for $i = 1\,..\,n$ and $\phi(w_1, \ldots, w_n)$. In cases where only the residue class of w_i modulo m is important, we may write the corresponding domain as $\mathbb{Z}_m$.

The predicate $\phi(w_1, \ldots, w_n)$ is given by a formula that is built up from "atoms" using arbitrary combinations of ANDs and ORs. An atom may express several types of relations among the w_i's: *(i) integer relations*, such as $F = 0$, $F \geq 0$, $F \equiv 0 \pmod m$, or $\gcd(F, m) = 1$, where F is an integer polynomial in the variables $w_1, \ldots, w_n$, and m is a positive integer; *(ii) group relations*, such as $\prod_{j=1}^{k} g_j^{F_j} = 1$, where the g_j's are elements of an abelian group, and the F_j's are integer polynomials in the variables $w_1, \ldots, w_n$; the descriptions of the groups appearing in such atoms will in general be given as system parameters (see 2.3); the group order need not be known, but certain technical restrictions apply.

It is known how to construct efficient protocols for these types of statements that, under reasonable assumptions, multi-realize $\mathcal{F}_{\text{ezk}}$ with joint access to a CRS. (As discussed in the full paper, we actually allow corrupt provers to submit witnesses lying in somewhat larger intervals; the ideal functionality has to be modified to allow for this.) The computational complexity of these proof systems can be easily related to the arithmetic circuit complexity of the polynomials that appear in the description of ϕ: the number of exponentiations is proportional to the sum of the circuit complexities; a more precise running time estimate depends on the types of groups and domains.

In some cases, we will write statements that quantify over certain variables using $\exists$ rather than $\nexists$. Roughly speaking, witnesses quantified under $\exists$ are asserted just to exist, rather than to be explicitly "known" by the prover. Making sense of this formally requires some effort; however, the effort pays off in that the resulting ZK protocols may be substantially more efficient.

6 Strong CAID/CAKE protocols

6.1 A protocol for vectored unions of product relations. We present here a ***protocol Π_0*** for $\mathcal{F}_{\text{caid}}^*$ that works for a vectored union of product relations (see §3.3).

We assume the relation is described by values $x_1, \ldots, x_k$ and $y_1, \ldots, y_k$. Party P has inputs $s_1 \in S_1^*, \ldots, s_k \in S_k^*$, and Q has inputs $t_1 \in T_1^*, \ldots, t_k \in T_k^*$. They are trying to determine if $\bigvee_{i=1}^{k} \big[(x_i, s_i) \in E_i \wedge (y_i, t_i) \in F_i\big]$, for fixed relations $E_1, \ldots, E_k$ and $F_1, \ldots, F_k$.

We assume that as system parameters, we have a group $\mathbb{G}$ of prime order q, and random generator g. We will need to assume that the computational Diffie-Hellman (CDH) assumption holds in this group. This protocol also requires some extra machinery, described below.

1a. *P computes $h_{\mathrm{L}} \leftarrow_{\mathrm{R}} \mathbb{G}$ and sends h_{L} to Q over a secure channel.*
1b. *Q computes $h_{\mathrm{R}} \leftarrow_{\mathrm{R}} \mathbb{G}$ and sends h_{R} to P over a secure channel.*
2a. *P waits for h_{R}, and then computes:*

$$\text{for } i = 1..k\text{: } \begin{cases} \alpha_i \leftarrow_{\mathrm{R}} \mathbb{Z}_q,\ \alpha'_i \leftarrow_{\mathrm{R}} \mathbb{Z}_q \\ \text{if } (x_i, s_i) \in E_i \text{ then } e_i \leftarrow g^{\alpha_i} \text{ else } e_i \leftarrow h_{\mathrm{R}}/g^{\alpha'_i} \end{cases}$$

Using $\mathcal{F}_{\mathrm{ezk}}$, P proves to Q:

$$\nabla\!\!\!\!\backslash \{s_i \in S_i^*, \alpha'_i \in \mathbb{Z}_q\}_{i=1}^k : \left[\bigwedge_{i=1}^k \Big((x_i, s_i) \in E_i \vee g^{\alpha'_i} = h_{\mathrm{R}}/e_i\Big)\right].$$

Note that $e_1, \ldots, e_k$ are delivered to Q via the $\mathcal{F}_{\mathrm{ezk}}$ functionality after P erases $\alpha'_1, \ldots, \alpha'_k$.
2b. *Q waits for h_L, and then computes:*

$$\text{for } i = 1..k\text{: } \begin{cases} \beta_i \leftarrow_{\mathrm{R}} \mathbb{Z}_q,\ \beta'_i \leftarrow_{\mathrm{R}} \mathbb{Z}_q \\ \text{if } (y_i, t_i) \in F_i \text{ then } f_i \leftarrow g^{\beta_i} \text{ else } f_i \leftarrow h_{\mathrm{L}}/g^{\beta'_i} \end{cases}$$

Using $\mathcal{F}_{\mathrm{ezk}}$, Q proves to P:

$$\nabla\!\!\!\!\backslash \{t_i \in T_i^*, \beta'_i \in \mathbb{Z}_q\}_{i=1}^k : \left[\bigwedge_{i=1}^k \Big((y_i, t_i) \in F_i \vee g^{\beta'_i} = h_{\mathrm{L}}/f_i\Big)\right].$$

Note that $f_1, \ldots, f_k$ are delivered to P via the $\mathcal{F}_{\mathrm{ezk}}$ functionality after Q erases $\beta'_1, \ldots, \beta'_k$.
3a. *P computes: for $i = 1..k$: if $(x_i, s_i) \in E_i$ then $u_i \leftarrow f_i^{\alpha_i}$ else $u_i \leftarrow_{\mathrm{R}} \mathbb{G}$*
3b. *Q computes: for $i = 1..k$: if $(y_i, t_i) \in F_i$ then $v_i \leftarrow e_i^{\beta_i}$ else $v_i \leftarrow_{\mathrm{R}} \mathbb{G}$*
4. *P and Q run a strong CAID subprotocol to evaluate the predicate $\bigvee_{i=1}^k (u_i = v_i)$, and output the result of this computation after erasing all local data.*

NOTES: *(i)* We have reduced our original strong CAID problem to a simpler strong CAID problem in Step 4. We discuss implementations of Step 4 below. *(ii)* The intuition for the main idea of the protocol runs as follows. Suppose, for example, that P is honest and Q is corrupt. In Step 2b, Q intuitively proves for each $i = 1..k$, either that it knows t_i such that $(y_i, t_i) \in E_i$ or that it *does not know* β_i; in the latter case, Q will not be able to predict the value $g^{\alpha_i \beta_i}$ when it comes to Step 4. *(iii)* Assuming the E_i's and F_i's are relations based on an anonymous credential system like IDEMIX, then all of the ZK protocols have relatively efficient implementations (see §5.1).

6.2 Security analysis. Our goal now is to show that protocol Π_0 realizes $\mathcal{F}_{\mathrm{caid}}^*$. Note that Π_0 is a hybrid protocol that uses the following ideal functionalities as subroutines: secure channels (i.e., $\mathcal{F}_{\mathrm{sch}}$), enhanced ZK (i.e., $\mathcal{F}_{\mathrm{ezk}}$) for relations of the form appearing in Steps 2a and 2b of the protocol, and $\mathcal{F}_{\mathrm{caid}}^*$ for relations of the form appearing in Step 4 of the protocol.

Theorem 4. *Under the CDH assumption for $\mathbb{G}$, protocol Π_0 realizes $\mathcal{F}_{\mathrm{caid}}^*$.*

6.3 Implementing Step 4. In the case where $k = 1$, one can use the equality test protocol in §7. As an alternative to protocol Π_0, in the case where $k = 1$ one can use a different protocol altogether, described in the full paper.

In the general case where $k \geq 1$, we suggest the following method. Assume we have a UC protocol for evaluating an arithmetic circuit mod N, where N is a system parameter that is the product of two large primes. Then to evaluate the boolean expression $\bigvee_{i=1}^{k}(u_i = v_i)$, P chooses $a_0 \in \mathbb{Z}_N$ at random, and for $i = 1..k$, encodes u_i as an element a_i of $\mathbb{Z}_N$; similarly, Q chooses $b_0 \in \mathbb{Z}_N$ at random, and for $i = 1..k$, encodes v_i as an element b_i of $\mathbb{Z}_N$. Then P and Q jointly evaluate in the expression $\prod_{i=0}^{k}(a_i - b_i)$ over $\mathbb{Z}_N$. If the boolean expression is true, then the expression over $\mathbb{Z}_N$ is zero; otherwise, the expression over $\mathbb{Z}_N$ evaluates to a random element of $\mathbb{Z}_N$.

Thus, we reduce the original strong CAID problem to a strong CAID problem for a simpler predicate, namely, boolean expressions of the form $\bigvee_{i=1}^{k}(u_i = v_i)$, and the latter is easily reduced to a simple circuit evaluation problem for expressions of the form $\prod_{i=0}^{k}(a_i - b_i)$ over $\mathbb{Z}_N$. There are quite practical protocols for circuit evaluation, which we discuss in detail in a companion paper. The basic idea is to use known techniques for circuit evaluation based on homomorphic encryption, making use of a semantically secure variant of Camenisch and Shoup's encryption scheme [6], which has the advantage that generating public keys is very inexpensive (making security with adaptive corruptions and erasures more practical) and proofs about plaintexts fit very nicely into the framework for ZK proofs discussed in §5.1. These protocols (and hence the resulting strong CAID protocols) require $O(k)$ exponentiations, and $O(\log k)$ (the circuit depth) rounds of communication, and $O(k)$ total communication complexity.

6.4 Adding key exchange. Adding key exchange is simple, especially since we are already assuming secure channels. We simply modify the protocol so that P generates a random key, and sends it to Q over a secure channel at the beginning of the protocol. In addition, whenever either party would output 1, it instead outputs $(1, K)$. This is a generic transformation that converts any $\mathcal{F}_{\text{caid}}^{*}$ protocol into an $\mathcal{F}_{\text{cake}}^{*}$ protocol. Some other variations of protocol Π_0, including one that deals with partial equality relations, are discussed in the full paper.

6.5 From strong CAID/CAKE to CAID/CAKE. We can instantiate protocol Π_0 to get a practical $\mathcal{F}_{\text{sch}}$-hybrid protocol Π_0' that multi-realizes $\mathcal{F}_{\text{caid}}^{*}$ (or any of the variations discussed above) with joint access to a CRS — the crucial building block is $\mathcal{F}_{\text{ezk}}$, discussed in §5. Then using the split functionalities techniques in §4, we can turn Π_0' into a protocol $\mathsf{s}\Pi_0'$ that multi-realizes $\mathsf{s}\mathcal{F}_{\text{caid}}^{*}$ with joint access to a CRS. The resulting protocol is a CAID/CAKE protocol that works without secure channels.

Typically, the purpose of running a CAKE protocol is to use the session key to implement a secure session. If, in fact, this is the goal, a more straightforward way of achieving it is as follows. Simply design a $\mathcal{F}_{\text{sch}}$-hybrid protocol that works as follows: first, it runs a strong CAID protocol, and if that succeeds, the parties continue to communicate, using the secure channels provided by the $\mathcal{F}_{\text{sch}}$

functionality. Now apply the split functionalities techniques in §4 to this protocol, obtaining a protocol that essentially provides a "credential authenticated secure channel".

7 A protocol for equality testing and a related problem

Here is a simple protocol for equality testing, called ***protocol $\Pi_{\mathbf{eq}}$***. We assume that a group $\mathbb{G}$ of prime order q, along with a generator $g \in \mathbb{G}$, are given as system parameters. We will need to assume the DDH for $\mathbb{G}$. We assume the inputs to the two parties are encoded as elements of $\mathbb{Z}_q$. Again, we use $\mathcal{F}_{\text{ezk}}$ as a subprotocol. The protocol runs as follows, where P has input $a \in \mathbb{Z}_q$, and Q has input $b \in \mathbb{Z}_q$:

1. *P computes: $h \leftarrow_{\text{R}} \mathbb{G}$, $x_1, x_2, r \leftarrow_{\text{R}} \mathbb{Z}_q$, $c \leftarrow g^{x_1} h^{x_2}$, $u_1 \leftarrow g^r$, $u_2 \leftarrow h^r$, $e \leftarrow g^a c^r$, and using $\mathcal{F}_{\text{ezk}}$ proves to Q: $\nabla a \in \mathbb{Z}_q \, \exists r \in \mathbb{Z}_q : g^r = u_1 \wedge h^r = u_2 \wedge g^a c^r = e$; note that h, c, u_1, u_2, e are delivered to Q via the $\mathcal{F}_{\text{ezk}}$ functionality after erasing r.*
2. *Q computes: $s \leftarrow_{\text{R}} \mathbb{Z}_q^*$, $t \leftarrow_{\text{R}} \mathbb{Z}_q$, $\tilde{u}_1 \leftarrow u_1^s g^t$, $\tilde{u}_2 \leftarrow u_2^s h^t$, $\tilde{e} \leftarrow e^s g^{-bs} c^t$, and using $\mathcal{F}_{\text{ezk}}$ proves to P: $\nabla b \in \mathbb{Z}_q \, \exists s, t \in \mathbb{Z}_q : u_1^s g^t = \tilde{u}_1 \wedge u_2^s h^t = \tilde{u}_2 \wedge e^s g^{-bs} c^t = \tilde{e} \wedge \gcd(s, q) = 1$; note that $\tilde{u}_1$, $\tilde{u}_2$, $\tilde{e}$ are delivered to P via the $\mathcal{F}_{\text{ezk}}$ functionality after erasing s, t.*
3. *P computes: $z \leftarrow_{\text{R}} \mathbb{Z}_q^*$, $d \leftarrow \tilde{e}^z (\tilde{u}_1)^{-zx_1} (\tilde{u}_2)^{-zx_2}$, and using $\mathcal{F}_{\text{ezk}}$ proves to Q: $\exists x_1, x_2, z \in \mathbb{Z}_q : g^{x_1} h^{x_2} = c \wedge \tilde{e}^z (\tilde{u}_1)^{-zx_1} (\tilde{u}_2)^{-zx_2} = d \wedge \gcd(z, q) = 1$; here, d is delivered to Q via the $\mathcal{F}_{\text{ezk}}$ functionality after erasing x_1, x_2, z.*
4. *After erasing all local data, both parties output 1 if $d = 1$, and output 0 otherwise.*

NOTES: *(i)* We are using $\exists$ as well as ∇ quantifiers here. This allows for certain optimizations, since values quantified under $\exists$ are never explicitly needed in the simulator in the security proof below, other than to verify the that the corresponding relation holds. *(ii)* In Step 1, (h, c) is the public key and (x_1, x_2) the private key for "Cramer-Shoup Ultra-Lite" — the semantically secure version of Cramer-Shoup encryption. (u_1, u_2, e) is an encryption of g^a. We will exploit the fact that this scheme is "receiver non-committing", as was demonstrated by Jarecki and Lysyanskaya [13]. This property will allow us to simulate adaptive corruptions. *(iii)* In Step 2, assuming (u_1, u_2, e) encrypts g^a, then $(\tilde{u}_1, \tilde{u}_2, \tilde{e})$ is a random encryption of $g^{s(a-b)}$. *(iv)* In Step 3, P is decrypting $(\tilde{u}_1, \tilde{u}_2, \tilde{e})$, and raising it to the power z, so that $d = g^{zs(a-b)}$ *(v)* All of the ZK protocols have practical implementations, as discussed in §5.1.

Theorem 5. *Assuming the DDH for $\mathbb{G}$, protocol Π_{eq} realizes functionality $\mathcal{F}_{\text{caid}}^*$ for the equality relation.*

Applications. One application of protocol Π_{eq} is in the implementation of Step 4 of protocol Π_0 (see §6.3). However, in this situation, a specialized protocol in the full paper is more efficient.

Another application is to PAKE protocols. We can efficiently implement $\mathcal{F}_{\text{ezk}}$ for the necessary relations using secure channels and a common reference string, and augment the protocol to share a random key over a secure channel. This gives us a fairly efficient strong CAKE protocol for the equality relation that uses secure channels. We then derive the split version of the protocol, using a simple

Diffie-Hellman key exchange as in §4, which realizes the CAKE functionality (more precisely, it multi-realizes the CAKE functionality with joint access to a CRS). As observed in [2], a protocol that realizes this functionality in fact realizes the PAKE functionality (as defined in [9]). Our particular protocol is probably a bit less efficient than the one in [9]; however, our protocol has the advantage of being secure against adaptive corruptions (assuming erasures). A very different PAKE protocol, with a structure similar to that in [9], that is secure against adaptive corruptions was recently presented in [1].

7.1 A variation. A variation on the above protocol gives a strong CAID protocol for the relation $DL := \{(a, g^a) : a \in \mathbb{Z}_q\}$. That is, it tests if $g^a = v$, where a is the input to P and v is the input to Q. The idea is to have Q "verifiably encrypt" v. The protocol, which we call ***protocol*** $\boldsymbol{\Pi_{\mathrm{dl}}}$, runs as follows:

1. *P computes: $h \leftarrow_{\mathrm{R}} \mathbb{G}$, $x_1, x_2, r \leftarrow_{\mathrm{R}} \mathbb{Z}_q$, $c \leftarrow g^{x_1} h^{x_2}$, $u_1 \leftarrow g^r$, $u_2 \leftarrow h^r$, $e \leftarrow g^a c^r$, and using $\mathcal{F}_{\mathrm{ezk}}$ proves to Q: $\aleph\, a \in \mathbb{Z}_q\, \exists\, r \in \mathbb{Z}_q : g^r = u_1 \wedge h^r = u_2 \wedge g^a c^r = e$; note that h, c, u_1, u_2, e are delivered to Q via the $\mathcal{F}_{\mathrm{ezk}}$ functionality after erasing r.*
2. *Q computes: $s \leftarrow_{\mathrm{R}} \mathbb{Z}_q^*$, $t \leftarrow_{\mathrm{R}} \mathbb{Z}_q$, $y \leftarrow_{\mathrm{R}} \mathbb{Z}_q$, $\tilde{v} \leftarrow g^y v$, $\tilde{u}_1 \leftarrow u_1^s g^t$, $\tilde{u}_2 \leftarrow u_2^s h^t$, $\tilde{e} \leftarrow e^s v^{-s} c^t$ and using $\mathcal{F}_{\mathrm{ezk}}$ proves to P: $\aleph\, y \in \mathbb{Z}_q\, \exists\, s, t \in \mathbb{Z}_q : u_1^s g^t = \tilde{u}_1 \wedge u_2^s h^t = \tilde{u}_2 \wedge e^s \tilde{v}^{-s} g^{ys} c^t = \tilde{e} \wedge \gcd(s, q) = 1$; note that $\tilde{v}, \tilde{u}_1, \tilde{u}_2, \tilde{e}$ are delivered to P via the $\mathcal{F}_{\mathrm{ezk}}$ functionality after erasing y, s, t.*
3. *P computes: $z \leftarrow_{\mathrm{R}} \mathbb{Z}_q^*$, $d \leftarrow \tilde{e}^z (\tilde{u}_1)^{-zx_1} (\tilde{u}_2)^{-zx_2}$, and using $\mathcal{F}_{\mathrm{ezk}}$ proves to Q: $\exists\, x_1, x_2, z \in \mathbb{Z}_q : g^{x_1} h^{x_2} = c \wedge \tilde{e}^z (\tilde{u}_1)^{-zx_1} (\tilde{u}_2)^{-zx_2} = d \wedge \gcd(z, q) = 1$; here, d is delivered to Q via the $\mathcal{F}_{\mathrm{ezk}}$ functionality after erasing x_1, x_2, z.*
4. *Both parties output 1 if $d = 1$, and output 0 otherwise.*

NOTES: *(i)* Step 1 is exactly the same as before. *(ii)* In Step 2, Q is generating a random encryption of $(g^a/v)^s$. Moreover, by giving $\tilde{v}$ and y to $\mathcal{F}_{\mathrm{ezk}}$, Q is effectively giving v to $\mathcal{F}_{\mathrm{ezk}}$. *(iii)* Step 3 is the same as before, but now $d = (g^a/v)^{sz}$.

Theorem 6. *Assuming the DDH for $\mathbb{G}$, protocol Π_{dl} realizes functionality $\mathcal{F}_{\mathrm{caid}}^*$ for the relation DL.*

Applications. This protocol, when augmented with a key sharing step over a secure channel, and "split" as in §4, gives us a practical PAKE protocol that is secure against adaptive corruptions *and server compromise*. That is, the client stores the password a, while the server stores g^a. If the password file on the server is compromised, then it will not be easy to an attacker to login to the server as the client.

Unlike previous protocols, such as in [11], our protocol does not rely on random oracles. To be fair, the definition of security in [11] is so strong that it probably cannot be achieved without random oracles: the security definition in [11] requires that in the event of a server compromise, an attacker must carry out an offline dictionary attack in order to guess the password. Also, note that the protocol in [11] is proved secure only in the static corruption model.

In a complete PAKE protocol, one would likely set $a := H(pw, clientID, serverID)$, where H is a cryptographic hash, pw is the

actual password, and *clientID* and *serverID* are the names of the client and server, respectively. If H is entropy preserving, pw is a high-entropy password, and the discrete logarithm problem in $\mathbb{G}$ is hard, then it will be infeasible to login as the client, even if the server is compromised. Moreover, if H is modeled as a random oracle, and the discrete logarithm problem in $\mathbb{G}$ is hard, then even in the event of a server compromise, an attacker must still carry out an offline dictionary attack in order to login as the client. Thus, our new protocol is the first fairly practical PAKE protocol (UC-secure or otherwise) that is secure against server compromise and does not rely on random oracles; as a bonus, it is also secure against adaptive corruptions.

References

1. M. Abdalla, C. Chevalier, and D. Pointcheval. Smooth projective hashing for conditionally extractable commitments. In *EUROCRYPT 2009*, pages 671–689, 2009.
2. B. Barak, R. Canetti, Y. Lindell, R. Pass, and T. Rabin. Secure computation without authentication. In *CRYPTO 2005*, pages 361–377, 2005. Full version at `http://eprint.iacr.org/2007/464`.
3. D. Beaver and S. Haber. Cryptographic protocols provably secure against dynamic adversaries. In *EUROCRYPT '92*, pages 307–323, 1992.
4. J. Camenisch, N. Casati, T. Gross, and V. Shoup. Credential authenticated identification and key exchange. Cryptology ePrint Archive, Report 2010/055, 2010. `http://eprint.iacr.org/`.
5. J. Camenisch and L. Lysyanskaya. Efficient non-transferable anonymous multi-show credential system with optional anonymity revocation. In *CRYPTO 2001*, pages 93–118, 2001.
6. J. Camenisch and V. Shoup. Practical verifiable encryption and decryption of discrete logarithms. In *CRYPTO 2003*, pages 126–144, 2003. Full version at `http://eprint.iacr.org/2002/161`.
7. R. Canetti. Universally composable security: a new paradigm for cryptographic protocols. Cryptology ePrint Archive, Report 2000/067 (December 14, 2005 version), 2005. `http://eprint.iacr.org`.
8. R. Canetti, Y. Dodis, R. Pass, and S. Walfish. Universally composable security with global setup. In *Theory of Cryptography 2007*, pages 61–85, 2007. Full version at `http://eprint.iacr.org/2006/432`.
9. R. Canetti, S. Halevi, J. Katz, Y. Lindell, and P. MacKenzie. Universally composable password-based key exchange. In *EUROCRYPT 2005*, pages 404–421, 2005.
10. R. Canetti and T. Rabin. Universal composition with joint state. In *CRYPTO 2003*, pages 265–281, 2003. Full version at `http://eprint.iacr.org/2002/047`.
11. C. Gentry, P. MacKenzie, and Z. Ramzan. A method for making password-based key exchange resilient to server compromise. In *CRYPTO 2006*, pages 142–159, 2006.
12. S. Jarecki, J. Kim, and G. Tsudik. Beyond secret handshakes: affiliation-hiding authenticated key agreement. In *RSA Conference, Cryptographers Track (CT-RSA 08)*, 2008.
13. S. Jarecki and A. Lysyanskaya. Adaptively secure threshold cryptography: introducing concurrency, removing erasures. In *EUROCRYPT 2000*, pages 221–242, 2000.