Have Your CKAKE and Eat it, Too: Efficient, Composable KEM-Authenticated Key Exchange

Myrto Arapinis, Christopher Battarbee, and Mina Doosti

School of Informatics, University of Edinburgh, United Kingdom

Abstract. We report on a novel authenticated key-exchange (AKE) protocol where the authentication is achieved entirely by key-encapsulation mechanisms (KEMs). Techniques to achieve AKE with KEMs have been known for some time, but have received renewed attention in a post-quantum world; in contrast to classical cryptography, the data corresponding to the NIST post-quantum KEM standard is a significant save on bandwidth compared to the signature standard. Previous KEM authenticated AKE protocols are not known to be composable; our protocol offers similar security guarantees, plus composability, while being more efficient in terms of bandwidth compared to non-composable KEM-based AKE protocols, and composable signature-based AKE protocols. Our protocol features a modular design, and a full security proof in the Constructive Cryptography (CC) framework, one of the major composable security frameworks. We also prove the forward secrecy of our protocol, and introduce generic techniques to prove forward secrecy in CC, which may be of independent interest.

1 Introduction

Authenticated key exchange (AKE) is a foundational primitive in the sphere of modern cryptography, and has been studied extensively. Informally, it deals with the task of negotiating a shared, secret value with another honest party, in the presence of a powerful adversary who controls all the communication between these parties. The vast majority of the literature, and in particular the associated security analysis, has been done in the game-based setting, which does not come with composability guarantees. This is a significant problem, from both a sequential, and parallel, point of view. Since AKE is explicitly designed to be sequentially composed – that is, the key which an AKE protocol establishes is useful only when composed when consumed by another protocol, usually some symmetric authenticated encryption scheme – we need to ensure that the key we establish remains secure under sequential composition. We are also frequently concerned with pairwise AKE between several honest parties (in this work we consider the common single-sever-many client setting), which means that instances of an AKE protocol will be running in parallel, all under the control of the same adversary. We therefore also have to worry about parallel composition.

At the same time, there has been recent attention on AKE in a post-quantum world. In 2024, NIST announced the first post-quantum standards, recommending a key encapsulation mechanism (KEM) and two digital signature schemes (DSS) as quantum-safe. These are known as ML-KEM, ML-DSA and SLH-DSA respectively. In contrast to classical cryptography, the KEM is significantly more space-efficient than either of the DSSs – ML-KEM ciphertexts are around around three times shorter than ML-DSA signatures with comparable public keys, and more than ten times shorter than SLH-DSA signatures, though this latter scheme has very short public keys. This poses a challenge from an AKE protocol design standpoint – the traditional design choice has been to establish authenticity with digital signatures, but can we develop techniques to authenticate entirely with KEMs and improve the efficiency of post-quantum AKE?

In fact, techniques in this direction have already been developed. In the work of Schwabe, Stebila and Wiggers in [SSW20; SSW21], a TLS-like protocol called KEM-TLS is put forward, and a security proof is given in the game-based setting – but the composable security of the protocol is not known. Coretti, Maurer and Tackmann show how to establish composably secure unilaterally authenticated keys with KEMs in [CMT13], but do not consider parallel protocol instances or forward secrecy. The same authors consider such AKE-related concerns in composable framework in [MTC13], but use signature authentication, and do not argue formally about forward secrecy.

In this work, we improve upon these results, offering a composably secure, KEM-authenticated, provably forward secret AKE protocol, which we call CK-AKE[1]. In line with existing work on the subject, security is proved in the Constructive Cryptography framework, one of the major composable security frameworks. Moreover, we do not sacrifice efficiency of the protocol in exchange for the stronger security guarantees; in fact, our protocol offers a moderate saving in bandwidth compared to KEM-TLS.

1.1 Contributions

We introduce a KEM-based AKE protocol and prove its security in the Constructive Cryptography framework, introducing generic techniques for handling forward secrecy and corruption in this framework that may be of independent interest. The security goes through with respect to the advantage in standard game-based definitions. Finally, we retain the efficiency of competitive, non-composable protocols.

1.2 Related Work

Bootstrapping the authenticity of public keys with cryptographic primitives ostensibly designed for confidentiality, rather than authenticity, is quite an old technique. The protocols in [Kra96; KW16] both use Diffie-Hellman key exchange with respect to static secrets, allowing for privacy and authenticity with

[1] Short for **C**omposable **K**EM-**A**uthenticated **K**ey **E**xchange, and pronounced 'cake'.

signatures – but NIST have not standardised any Diffie-Hellman like protocols and have instead focussed on KEMs. AKE based on generic KEMs is considered in [BCK98], where a generic encrypt-than-MAC approach that we also use is suggested; other examples include [Fuj+15; Hül+21; Boy+08]. The protocol we have mentioned in the introduction, KEM-TLS, [SSW20], is the most modern treatment of the subject and the one we most explicitly compare to. It differs from the other KEM-based protocols we have mentioned in that it considers a unilateral server-to-client scenario, and explicitly handles how the server's public key is distributed – we follow this ethos. On the other hand, KEM-TLS takes the TLS-like approach of carrying out an unauthenticated, ephemeral handshake and then using static-secret backed cryptography to check that parties' local copies of this handshake match, whereas we first establish authenticity, and only then carry out an ephemeral handshake – this fits better with the modular approach of constructive cryptography, and allows us to downgrade the security requirements of the KEM used for the ephemeral portion of the protocol. None of the prior work on KEM authentication considers composability.

The most famous and widely-used AKE protocol for unilateral, server-to-client authentication is TLS 1.3 [Res18], which uses digital signatures to authenticate an ephemeral handshake. In some sense, the 'obvious' way to bring this type of AKE into a post-quantum world is to replace the ephemeral Diffie-Hellman handshake with a KEM, and the classical signature with a post-quantum signature, so that we can appeal to the NIST standards. A full standard of a post-quantum version of TLS is ongoing work, but note that much of the focus has been on hybrid methods – that is, using post-quantum cryptography in tandem with classical cryptography, in such a way that one of the primitives can fail. Of course, as we have mentioned, continuing to use signatures for authentication incurs a heavy penalty on bandwidth with respect to KEM-based authentication. There has been some work on composable AKE, mostly in the Universal Composability (UC) framework [CG10; CG10], but these do not consider KEM authentication. In the CC framework, [CMT13] lays the groundwork for our protocol by showing that $\mathsf{ind}-\mathsf{cca}$ KEMs suffice to construct unilaterally authentic keys given a valid copy of the public key, but this paper does not consider these unilateral keys in a typical AKE scenario. Perhaps closest to the spirit of our work is the composable AKE protocol put forward in [MTC13]; we will invoke its ideas frequently, and our generic methods of handling forward secrecy, and saving bandwidth by securely hashing part of the transcript, can be seen as improvements on that work, independently of the interest in KEM authentication.

2 Preliminaries

2.1 Notation

Some standard notation is employed: $[n]$ means $\{1, ..., n\}$; S_n means the set of permutations of $[n]$. If a variable is to be assigned a value we use $\leftarrow$, reserving the $=$ symbol for checks of equality. The behaviour of the systems we discuss are

often parametrised by various control bits that set what 'mode' the system is in – we use q, r and s for these. The symmetric keys we are seeking to establish go by μ or ν. Ciphertexts, messages and authentication tags are always (possibly ornamented variants of) z, m and t; the spaces these values live in take calligraphic script (so the space of ciphertexts is $\mathcal{C}$, the space of messages is $\mathcal{M}$, and so on). An adversary can generally control communication in two ways; either by injecting its own choice of data over an insecure channel, or by triggering some corruption command and directly altering the internal state of the system: data provided by the former method is ornamented with a dash $\cdot'$, data provided by the latter is ornamented with a bar $\bar{\cdot}$. Finally, the systems often have outputs at the honest party sides that can take a range of values. We give these placeholder variables values b and c for notational convenience — an explicit value is assigned to this placeholder variable internally.

2.2 Constructive Cryptography

Note that preliminary material on Constructive Cryptography in the context of this paper can be found in [MTC13], authored by one of the developers of the framework.

We work in the Constructive, also known as Abstract, Cryptography (CC) framework [MR11]. This is a simulation-based framework and therefore has some natural composability guarantees. It differs, however, from the other major composable security framework (UC) in two ways: first, all the set-up assumptions (authentic copies of public keys, for example), as well as the desired ideal functionalities, are explicitly defined as *resources* (interactive, discrete and probabilistic systems, which different parties all connect to, but interact with in different ways). One can then 'attach' special stateless machines called 'converters' to these resources, such that the composition behaves identically to a different resource – this is the crucial notion of *construction*, where the field gets its name. Secondly, the ethos is inherently 'top-down', rather than 'bottom-up', in the following sense: in UC one might design a protocol, come up with ideal functionalities to describe each of its constituent primitives, and prove that the composition of these primitives is composably secure. In CC we start with the functionality we wish our protocol to achieve and show that various primitives (running inside converters) construct this functionality from the resources that represent our set-up assumptions. On the other hand, the modern treatment of AKE endows the protocols of interest with various advanced properties, such as forward secrecy. In contrast with the work on UC, CC does not come with the type of 'corrupt' interfaces native to UC ideal functionalities. Indeed, we are not aware of any CC treatment of AKE that models static secret compromise.

In this section we give an overview of resources, converters, construction, composition, and finally the proof tool of reduction.

Resources. A resource is an interactive random system with various interfaces, each of which may be accessed by some party (several interfaces can be accessed by the same party). At these interfaces the various parties can make discrete

queries and receive (possibly probabilistically generated) responses, where the distribution of any particular output variable may depend on previous queries at that, and other, interfaces. The resources may also generate some internal state which they keep track of. Examples of resources include all communication channels, and key distribution resources.

Converters. We model the application of a protocol to an assumed resource by attaching systems to the interfaces of resources, called converters. These have exactly two interfaces; one (the *inner* interface) which interfaces with data from the resource to which it is attached, and one (the *outer* interface) which outputs data to the relevant party. Each of these interfaces can be divided into *sub*-interfaces to deal with the presence of a resource having multiple interfaces corresponding to a particular party. One advantage of CC is not having to specify the computation model - nevertheless, we can think of converters as having Turing machines aboard.

Single Server, Many Clients. Each of the resources we describe model the single-server-many-client scenario; whereby there is one party called the *server*, who communicates with n clients (the clients do not communicate with each other). We model this scenario by defining our initial resource to have n interfaces for the server, and n for the clients, plus an extra one to handle adversarial capabilities. To the n server interfaces, which we call the *server side*, a single converter is attached[2], where the server keeps track of data from all the concurrent sessions. We attach one converter each to each of the n client interfaces (the *client side*), so that each converter is keeping track only of a single protocol. Note that a converter is defined to have only a single outer interface, but we would still like to logically separate input/output data from the server's converter. As such we will talk of a *client interface*, but a *server sub-interface* until the sessions are separated, whereupon we can talk about client and server interfaces in parallel.

Labelling and Re-routing. Since every client is assured of the identity of the server, and we do not aim to offer the server any guarantees as to the identities of each of the clients, there is no canonical labelling of the clients and server sub-interfaces; we only care that each server sub-interface corresponds to exactly one client (such a labelling is said to be *bijective*). As such any bijective labelling we fix is arbitrary, and another arbitrary labelling can be trivially forced by the adversary by re-routing data over insecure channels. We handle this by assuming there is an arbitrary, but fixed, labelling of the n clients and of the n server sub-interfaces; the resources allow the input of a permutation that determines which sub-interface is bound to which client with respect to this labelling.

[2] Since converters have a single inner interface, and receiving data from the resource is only one of the functions a particular converter might perform, we can have not only sub-interfaces but sub-sub-interfaces.

Construction. Suppose there is a resource A to which we have attached converters $\{\pi\}_i$. We say that these converters ϵ-*construct* a resource B from A (equivalently, the composite system ϵ-*realises* B) if there is a *simulator* that attaches to B such that any *distinguisher* cannot guess if it is interacting with A and its converters, or B and its simulator, except with probability negligibly close to 1/2. We call this probability the *distinguishing advantage*. Here:

- A *simulator* is a special type of converter that usually attaches at the adversary's interfaces, and uses B to simulate A (note that the two candidate systems do not necessarily have the same syntax, which the simulator will have to handle)
- A *distinguisher* is a system which attaches to all the interfaces of the candidate system, and after making its choice of queries to the system outputs its guess of which system it is interacting with

Note that since we bound the distinguishing advantage by the advantage of an adversary in a security game, and this advantage is a function of a security parameter (rather than a concrete value), the distinguishing advantage itself will often be a function of a security parameter.

We will call A and its converters the "real world" or "real system"; similarly B and the simulator are called the "ideal world", or "ideal system". The intuition behind the security is that if no distinguisher can guess which world it is in, then B can be said to simulate the real world; in particular, any attack that can happen in the real world can happen in the ideal world, and so the system we construct by attaching protocols to A is at least as 'secure' as the guarantees of resource B.

The canonical example of this approach is to assume we have a resource that distributes secret keys, and an authenticated channel (we will come to the formal definition of these later; roughly, they are communications channels that offer a 'read-only' capability to an adversary). To these resources attach converters implementing the one-time pad, using the key distributed to them by the assumed resource. One can show that this composite system constructs a secure channel (a communication channel that reveals only the length of the message passed down it) in the presence of an unbounded distinguisher.

Composition. There are two, equally important, notions of composition: *parallel*, and *sequential*. The first deals with the security of multiple concurrent instances of a protocol: the parallel composition of resources is just the two resources with their interfaces merged, and similarly for the parallel composition of interfaces. For two resources A and B, and two converters π and π', denote their parallel composition $\mathsf{A} \otimes \mathsf{B}$. Sequential composition, meanwhile, describes the behaviour of systems as more and more complex converters are attached to them: there is no notion of the sequential composition of converters, and converters are sequentially composed by attaching the inner interface of one to the outer interface of another. For two converters π and π', we write $\pi' \circ \pi$ for their sequential composition, where π is understood to be the innermost of the two converters in this arrangement. Note that sequential composition is rather

more subtle than parallel composition. The output from an outer interface of the innermost converter will not necessarily comprise the full set of data that the outermost converter needs as input - for example, the outermost converter in this work is often expecting additional, adversarially controlled input via an insecure channel. Moreover, two converters do not necessarily have a well-defined notion of sequential composition. In general, we need to be careful to define explicitly how interfaces are connected.

There are then two composition results:

- **[Parallel Composition]**. If converters π_1 and π_2 attached to resource A_1 ϵ_1-construct resource B_1, and converters ψ_1 and ψ_2 attached to resource A_2 ϵ_2-construct resource B_2, then converters $\pi_1||\psi_1$ and $\pi_2||\psi_2$ attached to $\mathsf{A}_1||\mathsf{A}_2$ $(\epsilon_1 + \epsilon_2)$-construct $\mathsf{B}_1||\mathsf{B}_2$.
- **[Sequential Composition]**. If converters $\pi_1, ..., \pi_n$ attached to resource A ϵ_1-construct resource B, and converters $\psi_1, ...\psi_n$ attached to resource B ϵ_2-construct resource C, then converters $\psi_1 \circ \pi_1, ..., \psi_n \circ \psi_n$ attached to resource A $(\epsilon_1 + \epsilon_2)$-construct resource C.

Games and Reductions We are going to construct the various ideal resources seen throughout this paper with standard cryptographic protocols. Ideally, we would relate the distinguishing advantage to the standard, game-based security definitions of these protocols, so that the composability of these protocols goes through with respect to standard definitions, rather than ad hoc security properties that are less well studied.

We can achieve this via the notion of a reduction. A reduction acts like a distinguisher in that it is part of the 'environment' and allowed to interact with arbitrary interfaces; we arrange such reductions in two ways, depending on the type of game.

Distinguishing games can be ported to our setting quite naturally; the reduction sits between the game and the distinguisher, using the game's oracles to respond to inputs from the distinguisher, and returning game outputs to the distinguisher. From the point of view of the game, the combined reduction and distinguisher behave like an adversary playing that game; and so the distinguishing advantage is bounded by the success probability of this adversary.

Other security games, whereby the validity of some adversary-submitted value is tested, are more subtle to handle. In this case the reduction connects to all the outer adversaries of the distinguisher (so the reduction is outside of the distinguisher, rather than inside as in the previous case). We can then appeal to a version of the famous identical-til-bad technique of Bellare and Rogaway [BR06] developed by Maurer in [Mau13, Lemma 2]. The reduction keeps track of whether the transcript of interaction with the distinguisher defines a winning event in the security game the adversary is playing; if we have that the real system and ideal system behave identically unless the game is won, the distinguishing advantage is bounded by the adversary's advantage in that security game.

2.3 The Resources

Per the constructive approach we describe the various resources that appear in this work. Recalling that real-world protocols construct some ideal functionality from another ideal functionality, it will also be appropriate to justify some of the modelling choices implicit in these definitions.

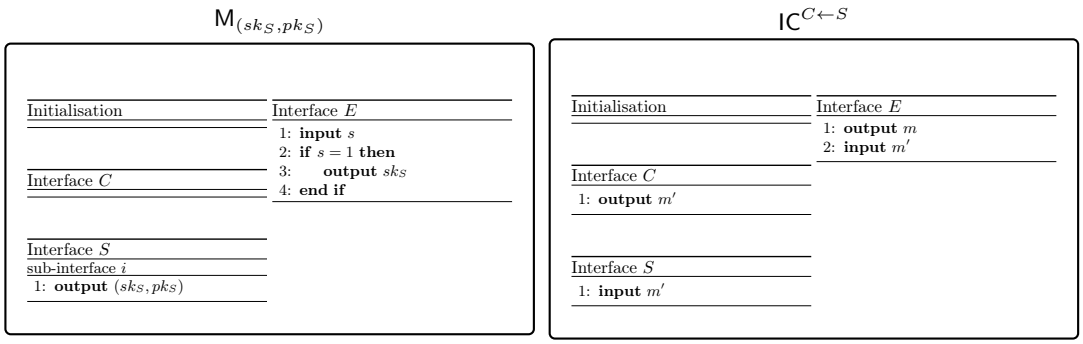


Fig. 1: The constructed resources: an n-party unilateral key resource (top left), an authenticated channel (top right), a weak two-way authenticated channel (bottom left), and an explicitly unilateral key resource (bottom right).

Assumed Resources

Authentic Public Keys Intuitively, a one-way authenticated channel is a resource with an interface for each of the honest parties, and an adversarial interface. The sender inputs messages to the resource, which are output at at the receiver's interface and the adversary's interface; in particular the receiver is guaranteed that a message output at its interface was input by the sender at its interface.

Both of [MTC13; CMT13] assume an ideal one-time authenticated channel to model that the server's public key can be transmitted to clients without tampering; say via the transmission of a certificate, or by its inclusion on hardware. On the other hand, the technique (which we adapt) of establishing an ephemeral key in [MTC13] relies on constructing a one-time, server-to-client authenticated channel, so we have a problem: it is rather circular to argue that an initial one-time authenticated channel constructs the one-time authenticated channel needed for a KEM exchange.

This apparent contradiction has to do with how we model public-key infrastructure in the CC framework – essentially, the set-up authenticated channels need only to be able to authentically relay the server's public key, but the authenticated channels we construct require the ability to authentically relay an arbitrary message. Indeed, standard AKE protocols implicitly think of a digital

certificate as a resource parametrised by the static public key that relays only that static public key – a token, of sorts, that uses the transferable verifiability of signatures to allow authentic relay of a particular public key between parties who possess an authentic copy of a certificate authority's public key. In principle we could replace our setup ideal authenticated channel with a resource modelling this functionality, but leave this for future work for the following reason: there does not exist in the literature a constructive treatment of a protocol realising some kind of public key infrastructure, and it is not clear precisely which resource would be realised by such a protocol. For the time being, then, we follow the existing tradition in the literature of considering the setup ideal authenticated channel to be a slightly crude model of public key infrastructure.

Insecure Channel IC. This is the standard communication channel with no guarantees: any input by an honest party is output at the adversarial interface, and any output was input at the adversarial interface. Such a model is in line with the Bellare-Rogaway tradition of adversaries capable of controlling all communication, including honest communication. We always assume that insecure communication is available, and as such this type of resource is 'free', in the sense that if resource B is constructed from resource A and an insecure channel, we can just as well say that resource B is constructed from resource A.

Each stage of the construction requires n parallel insecure channels, which are either server-to-client, or vice versa. In the former case, from the adversary's point of view each insecure channel yields the data input at a particular server sub-interface, and the label arbitrarily assigned to that sub-interface; meanwhile, inputs to each of the insecure channels can be understood as addressed to a client with a particular label. A server-to-client variant of the insecure channel is defined in Figure 1; the client-to-server variant is defined symmetrically.

Memory Cells M_{sk_S,pk_S}. In order to argue about forward secrecy we follow the formalism of [Jos20], and imagine that static secret keys are stored in a resource, rather than generated locally by a protocol. The static key pair contained in the memory cell is considered as an input to the protocol, rather than being generated locally by any component, so we consider the resource to be parametrised by a particular key pair. In this way we can model corruption – which is not possible if static keys are generated locally by a protocol, because converters are defined only to have a single interface attached to a resource, rather than to the interface of another party – and the common practice of storing static keys on distinct hardware (such as a Hardware Security Module). The resource is defined in Figure 1.

Constructed Resources.

Protocols in Parallel. We might hope, in general, to be able to consider our security analysis in a two honest party case (sometimes called the $\{A, B, E\}$ setting), and then use the parallel composition theorem to conclude that we have

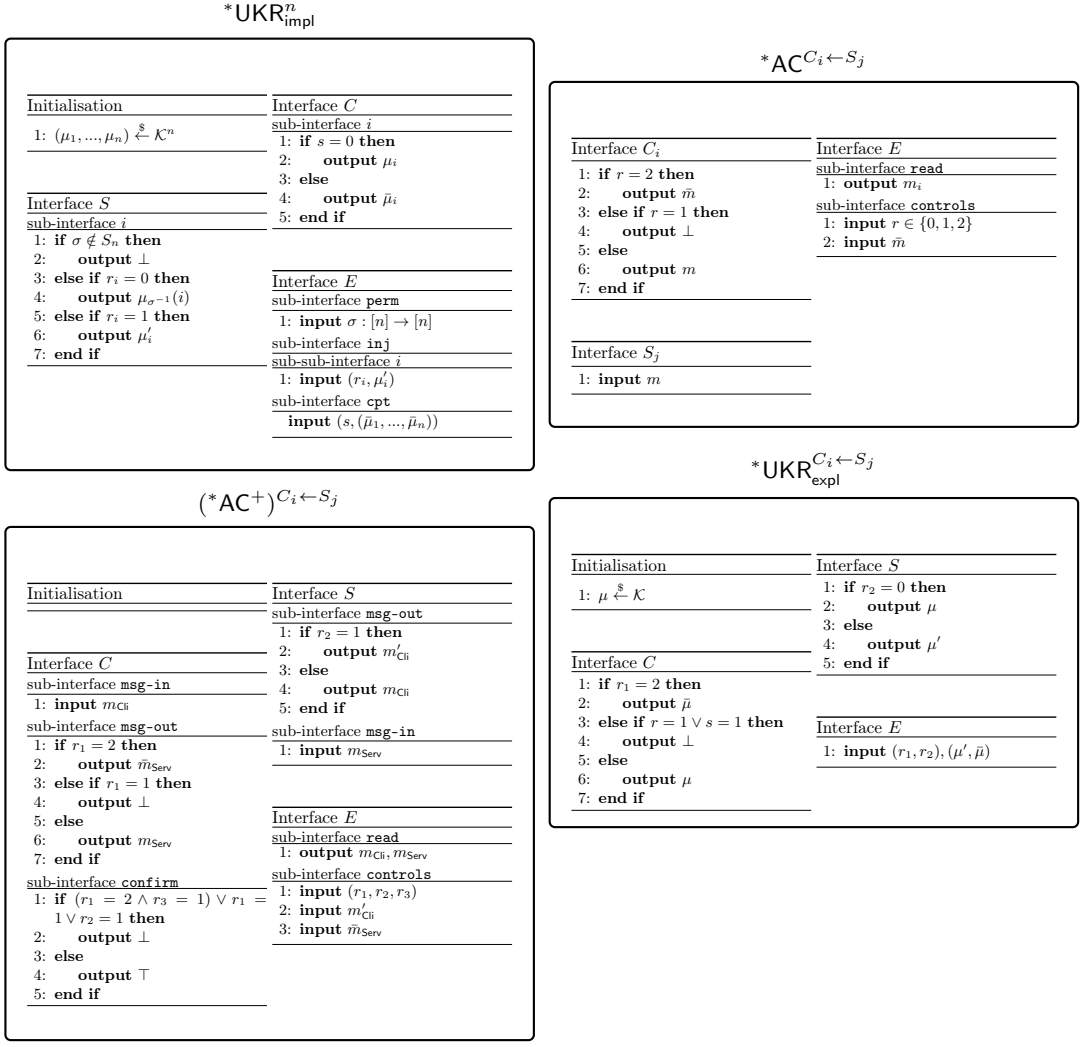


Fig. 2: The constructed resources.

security when several server-client sessions running in parallel. This is unfortunately not always possible, at least with our current formulation of parallel composition. The problem has to do with the shared state between protocol executions: one can demonstrate a distinguishing strategy between two protocols using the same public key, and the parallel composition of ideal functionalities, that succeeds with probability one. When we construct unilateral keys with n KEMs encapsulated against the same public key, therefore, we are are forced to argue directly about a 'multi-party' case, defining an ideal resource that explicitly deals with multiple clients. As such the first of the following definitions is unavoidably rather inscrutable, but the intuition for the more involved parts of the definitions should be understood as representing additional hard-coding that handles difficulties with parallel composition[3], especially the ability of an adversary to re-route communication.

Once we build authenticated channels on top of the unilateral keys, remarkably, the sessions become separated automatically. Each client will only verify with respect to a locally generated secret, and we can easily prohibit duplication of secrets on the server side; in other words, the re-routing capabilities of an adversary causes the construction of parallel authenticated channels, up to a permutation specified by the adversary. From here we can argue about resources in a simpler $\{A, B, E\}$ setting, and apply parallel composition.

(Corruptible) Implicitly Unilateral n-Key Resource ${}^{*}\mathsf{UKR}^{n}_{\mathsf{impl}}$. We construct a resource based on the unilateral key resource described in [CMT13], but with two extra features: we cater to one server and n clients, and we add a corruption capability to model for the corruption of the server's long-term secrets. The resource generates n keys uniformly at random internally; each client gets one of these keys, and the server gets n keys (one at each sub-interface) with no guarantees as to their provenance. To model the potential for re-routing over insecure channels, we allow the adversary to select at which server sub-interface it wishes each client's secret to appear. We also allow the adversary to deliver its own choice of secret at any sub-interface. Not counting those interfaces for which the adversary has chosen to inject its own secret, we insist that the re-routing defines a permutation (which is easy to enforce in the real world by having the server's convertor reject any vector of plaintexts that are not pairwise distinct, at the cost of some birthday bound in the advantage). This will be useful when we wish to separate sessions for higher functionalities. Finally, we include a corrupt functionality; if the corrupt marker is set to 1 the adversary can also select keys to be output at the client side. For the formal definition, see Figure 2.

Notice that the term 'unilateral key resource' is used in [CMT13] and [MTC13], but does *not* refer to the same thing in both cases: in the former an *implicitly*

[3] Issues with parallel composition of this kind appear to be endemic to the field. Closest to the spirit of our work, notice that the same thing happens in the signature-authenticated protocol of [MTC13] – even though a signature constructs an authenticated channel in the two-party case, across parallel sessions one constructs the 'bulletin board' functionality, which is distinct from the parallel composition of authenticated channels.

unilateral key resource is constructed, in the latter an *explicitly* unilateral key resource is constructed. The difference is that in the implicit case, the client always gets a key, even if the server gets an adversary-injected key; in the explicit case the client gets an error when the adversary injects a key. The distinction is a subtlety pertaining to KEM authentication that has already been pointed out in the proof of the KEM-TLS protocol [SSW20].

We can describe the next two resources in a less cumbersome fashion, since we will realise parallel compositions of A, B, E-setting resources. However, because the adversary can choose the registration of clients and sub-interfaces, they can also choose how the parallel composition is 'wired'. Per our discussion in Section 2.3, this is not a security issue.

(Corruptible) Authenticated Channel ${}^*\mathsf{AC}^{C_i \leftarrow S_j}$. By attaching message authentication protocols to the implicitly unilateral n-key resource we can obtain n separated authenticated channels in parallel. However, we cannot suppress the re-routing capabilities an adversary has when interacting with the unilateral n-key resource, and so for any server sub-interface, the adversary will be able to choose the client that will receive messages input to that sub-interface. Crucially, however, the adversary is restricted to choosing exactly one such client for each sub-interface. Again, as we have discussed, from the perspective of unilateral key exchange we only care that sessions are properly separated because we are guarantees on client identities are not in scope. We point out that the 'parallel composition of authenticated channels' constructed in [MTC13] is the same resource, though the ability of an adversary to select the wiring is framed less explicitly.

In Figure 2 we describe one of the authenticated channels that will be considered in parallel, which accepts message inputs at a server sub-interface S_j, and outputs messages at a client interface C_i. The resource ${}^*\mathsf{AC}^n$ is the parallel composition of n of these resources with respect to an adversarial-selected permutation σ; that is,

$$ {}^*\mathsf{AC}^n = \bigotimes_{i \in [n]} {}^*\mathsf{AC}^{C_i \leftarrow S_{\sigma(i)}} $$

We can think of this resource as being initialised by submitting a permutation σ, whereupon it will behave like parallel authenticated channels according to the 'rewiring' implied by σ.

Each of these channels is built from a corruptible symmetric key, and so themselves can be corrupted – where corruption in this case means causing a message of the distinguisher's choice to be output at the appropriate client interface.

(Corruptible) Weak Two-way Authenticated Channel $({}^*\mathsf{AC}^+)^{C_i \leftarrow S_j}$. One can use an authenticated channel from the server to the client to construct a weak authenticated channel in the other direction. In practical terms, when the client sends an ephemeral public key to the server, we require that that the server

responds with an encapsulation against the public key it received and a hash of that public key – otherwise, we have know way of knowing that the correct public key was encapsulated against. It was not previously known that a hash of the public key suffices (the protocol in [MTC13] requires the whole public key to be sent, which is obviously a significant drawback in terms of bandwidth); and the security analysis demonstrating this improvement is rather delicate. We handle it with the introduction of an intermediary resource, called (AC^+); again, we are dealing with n of these channels in parallel, which are subject to an initial choice of 'rewiring' by the distinguisher.

We have a notion of corruption, corresponding to the corruption of the underlying server-to-client authenticated channel. In this case the corruption is necessarily somewhat convoluted; once corruption occurs the behaviour of the system must be specified by an extra variable, the value of which is only relevant in the case of corruption.

Similarly to the previous case we consider the parallel composition of these channels with respect to some adversary-defined permutation σ, and construct the following.

$$({}^*\mathsf{AC}^+)^n = \bigotimes_{i\in[n]} ({}^*\mathsf{AC}^+)^{C_i \leftarrow S_{\sigma(i)}}$$

(Corruptible) Explicitly Unilateral Key Resource ${}^*\mathsf{UKR}_{\mathsf{expl}}^{C_i \leftarrow S_j}$. Finally, by carrying out an ephemeral key exchange protocol over the $({}^*\mathsf{AC}^+)^n$ channels we have established, we obtain *explicitly* unilateral keys, described in Figure 2. The difference between the implicit and explicit case is that in the former, the client always gets a key, regardless of whether the server has a legitimate key; in the latter, the client only gets a key if the server does not share their key with the adversary.

Compared to the unilateral key resource, more pertinent than the upgrade to an explicitly unilateral key is the introduction of forward secrecy. This is essentially a property of the underlying authenticated channels: because corruption will now allow knowledge of a key used solely for authentication, and we use this authenticated channel to establish a fresh and independent key, corruption will have no effect provided it occurs after the delivery of a ciphertext on this authenticated channel. We discuss forward secrecy in Section 3.1.

As in the previous two cases, we construct a parallel composition of these resources with respect to an adversary-selected permutation σ, that is:

$${}^*\mathsf{UKR}_{\mathsf{expl}}^n = \overset{*}{\bigotimes_{i\in[n]}} \mathsf{UKR}_{\mathsf{expl}}^{C_i \leftarrow S_{\sigma(i)}}$$

2.4 Cryptographic Definitions

Our results show that a protocol consisting of standard cryptographic primitives is composably secure, and that this composability goes through with respect to

the advantage in the usual KEM game. As such we now define various primitives, and their respective security games.

Universal Hash Functions Introduced by Wegman and Carter [CW77], universal hash functions define unconditionally secure one-time authentication (among myriad other non-security related applications). They are defined as follows.

Definition 1. *Let $\mathcal{H}$ be a family of functions from $\mathcal{M}$ to $\mathcal{T}$. For $0 < \varepsilon < 1$, suppose we have:*

1. *For every $x \in \mathcal{M}, t \in \mathcal{T}$, one has*

$$|\{h \in \mathcal{H} : h(x) = t\}| = \frac{|\mathcal{H}|}{|T|}$$

2. *For every $x_1, x_2 \in \mathcal{M}, t_1, t_2 \in \mathcal{T}$,*

$$|\{h \in \mathcal{H} : h(x_1) = t_1, h(x_2) = t_2\}| \leq \frac{\epsilon|\mathcal{H}|}{|T|}$$

Then $\mathcal{H}$ is called an ϵ-strongly universal family of hash functions[4]*.*

In [WC81], only $\epsilon = 1/|B|$ is defined. Indeed, this value is optimal, and if it is obtained one checks the second condition implies the first condition. The relaxation of the second condition to include non-optimal bounds, in which case the first condition is not immediate, is Stinson's treatment [Sti94].

Key Encapsulation Mechanisms A Key Encapsulation Mechanism, or KEM, is closely related to the notion of public-key encryption. Intuitively, a KEM can be understood as the result of generating a random value for use as a key, then encrypting this random value with respect to a PKE scheme. On the other hand, not all KEMs arise in this way, and so a separate treatment is given here for the sake of clarity.

Definition 2. *A KEM Ψ is a trio of algorithms* KeyGen, Enc, Dec. *Associated to these algorithms are a an asymmetric key space $\mathcal{K}_s \times \mathcal{K}_p$, a symmetric key space $\mathcal{K}$, and an encapsulation space $\mathcal{C}$, consisting of all the admissible private-public key pairs, symmetric keys, and encapsulations, respectively. The algorithms are defined as follows:*

- KeyGen *takes a security parameter κ as an input, and outputs a key pair (sk, pk) (sometimes called the 'secret key' and the 'public key', respectively)*
- Enc *is a non-deterministic algorithm that takes as input a public key $pk \in \mathcal{K}_p$, and outputs a symmetric key τ sampled uniformly at random from $\mathcal{K}$, and an encapsulation $z \in \mathcal{C}$.*

[4] Notice that unlike the more common modern cryptographic notion of a hash function, these particular hash functions have fixed-length inputs.

– Dec *takes as input a secret key* $sk \in \mathcal{K}_s$ *and an encapsulation* z*, and outputs either* $\perp$ *(error), or a symmetric key* τ*.*

The security of KEMs is defined such that encapsulations should be in some sense independent of the underlying plaintexts. We also need multiple parallel instances of the usual game, which has to do with the issues with parallel composition we have briefly discussed above.

Definition 3. *Let* $\Psi = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *be a* KEM*, and let* $\mathsf{Exp}^{\mathsf{n-ind-cca}}_{\Psi,\mathcal{A}}(\kappa)$ *be the experiment described in Figure 3. The advantage of* $\mathcal{A}$ *in the* $\mathsf{n-ind-cca}$*-indistinguishability game against* Ψ*, denoted* $\mathsf{Adv}^{\mathsf{n-ind-cca}}_{\Psi,\mathcal{A}}(\kappa)$*, is a function of the security parameter defined by*

$$\mathsf{Adv}^{\mathsf{n-ind-cca}}_{\Psi,\mathcal{A}}(\kappa) = \mathbb{P}\left(\mathsf{Exp}^{\mathsf{n-ind-cca}}_{\Psi,\mathcal{A}}(\kappa) = 1\right)$$

Game $\mathsf{Exp}^{\mathsf{n-ind-cca}}_{\Psi,\mathcal{A}}(\kappa)$

$(pk, sk) \leftarrow \mathsf{KeyGen}(\kappa)$
$\{(z_i^*, \mu_{0,i})\}_{i=1,\ldots,n} \leftarrow \mathsf{Enc}(pk)^{\otimes i}$
$(\mu_{1,1}, ..., \mu_{1,n}) \xleftarrow{\$} \mathcal{K}^{\otimes i}$
$b \xleftarrow{\$} \{0,1\}$
$\mathbf{z} \leftarrow (z_1^*, ..., z_n^*)$
$\mu_b^* \leftarrow (\mu_{b,1}, ..., \mu_{b,n})$
$b' \leftarrow \mathcal{A}^{\mathsf{Dec}()}(\tilde{z}, \tilde{\mu_b}, pk)$
return $b' \stackrel{?}{=} b$

Oracle $\mathsf{Dec}(z)$

if $z \neq \tilde{z}$ **then**
  return $\mathsf{Dec}(sk, z)$
else
  return $\perp$
end if

Fig. 3: The KEM indistinguishability game in the presence of chosen-ciphertext attacks.

It is well-known that $\mathsf{Adv}^{\mathsf{n-ind-cca}}_{\Psi,\mathcal{A}}(\kappa) \leq n \cdot \mathsf{Adv}^{\mathsf{1-ind-cca}}_{\Psi,\mathcal{A}}(\kappa)$, where this latter quantity represents the standard notion of $\mathsf{ind-cca}$ security, and is usually just written $\mathsf{Adv}^{\mathsf{ind-cca}}_{\Psi,\mathcal{A}}(\kappa)$.

If we do not allow the adversary access to the Dec oracle we instead get $\mathsf{ind-cpa}$ security – noting that since the public key is provided in the experiment, the adversary can always compute encapsulations of its choice. This is a strictly weaker requirement than $\mathsf{ind-cca}$ security, in the sense that one can exhibit KEMs that satisfy the former, but not the latter, security notion. From a practical perspective, a common technique for obtaining an $\mathsf{ind-cca\ kem}$ is to pass an $\mathsf{ind-cpa\ KEM}$ through a compiler called the Fujisaki-Okamoto transform [FO99], which induces a computational overhead. Therefore, if it is possible to realise a security guarantee with only $\mathsf{ind-cpa}$ security, it is desirable to do so from an efficiency standpoint.

For syntactic reasons we require that every $c \in \mathcal{C}$ decapsulates to a value in $\mathcal{K}$ (rather than returning an error, as is sometimes the case). We could drop this assumption at the cost of some minor syntactic patching of the proof, but since the assumption is true of ML-KEM we keep it in.

Cryptographic Hash Functions Despite the unfortunate overlap in terminology, *cryptographic* hash functions serve a different purpose for us than *universal* hash functions. They are defined as follows.

Definition 4. *A function H that takes arbitrary bitstrings in $\{0,1\}^*$ to a fixed tag space $\mathcal{T}$ is called a cryptographic hash function.*

The security guarantees are computational, rather than unconditional, and come in three distinct flavours. Only one of these is germane to our analysis.

Game $\mathsf{Exp}^{\mathsf{spr}}_{H}, \mathcal{A}$

1: Pick m
2: $h \leftarrow H(m)$
3: $m' \leftarrow \mathcal{A}(H, m)$
4: **if** $(m' \neq m) \wedge (H(m') = h)$ **then**
5: **return** 1
6: **else**
7: **return** 0
8: **end if**

Fig. 4: The second pre-image resistance game for a cryptographic hash function.

Definition 5. *Let H be a cryptographic hash function, and let $\mathsf{Exp}^{\mathsf{spr}}_{H,\mathsf{A}}$ be the experiment described in Figure 4. The advantage of $\mathcal{A}$ in the second pre-image resistance game against H, denoted $\mathsf{Adv}^{\mathsf{spr}}_{H,\mathcal{A}}$, is a — defined by*

$$\mathsf{Adv}^{\mathsf{spr}}_{H,\mathcal{A}} = \mathbb{P}\left(\mathsf{Exp}^{\mathsf{spr}}_{H,\mathsf{A}}\right)$$

3 The Protocol

We begin by giving a traditional overview of the full protocol. In fact, this 'full' version of the protocol is the sequential composition of 3 protocols; later we will prove the security of each of these in turn, and apply the composition theorem.

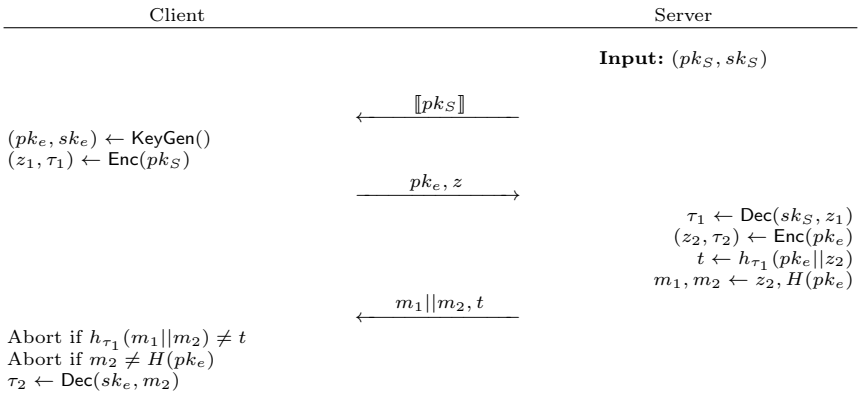


Fig. 5: The composable KEM-based AKE protocol. Note that the static key-pair (pk_S, sk_S) is not generated locally by the server, but is pre-generated and considered an input to the protocol. The $[\![\cdot]\!]$ notation denotes the requirement that pk_S be delivered to the client unaltered – this could be achieved by the transmission of a certificate, or pre-installation of a public key on the clients' browser. It is assumed that the other transmissions are controlled by the adversary.

The protocol works as follows: the client encapsulates against the server's static public key, generates an ephemeral key pair (pk_e, sk_e) locally, and sends the resulting ciphertext and its public key over an insecure channel. The server encapsulates against this ephemeral public key – the corresponding key will be the output of the protocol. The key that was encapsulated against the server's public key, meanwhile, is consumed in the authentication, via a universal hash function, of the ephemeral handshake that establishes this output key.

3.1 Security Proof

The aim of this section is to prove the following main theorem:

Theorem 1 (Informal.). *The protocol in Figure 5 yields n composably secure unilateral keys between a server and n clients. If the server's private key is corrupted, any keys established before this corruption remain secure.*

As discussed, we demonstrate that simpler protocols construct progressively more advanced resources, until we arrive at the target resource. By the sequential composition theorem, the composition of these protocols construct the final ideal resource from the assumed initial resources, with a distinguishing advantage not exceeding the sum of the distinguishing advantages corresponding to the intermediate proof steps. One can see this as analogous to the game-hopping technique common in game-based proofs of security; a similar technique was deployed in a constructive treatment of TLS 1.3 [Koh+15]. As discussed in Section 2.3, we will have to apply sequential composition to resources explicitly defined to cater to a multi-party setting.

We proceed in three main intermediate steps: constructing an implicitly unilateral key resource from authentic copies of a public key; constructing one-time authenticated channels from an implicitly unilateral key resource; and constructing an explicitly unilateral key resource from authenticated channels. The composition of these protocols gives us the full AKE protocol, and so we conclude that we construct an explicitly unilateral key resource from a one-time authenticated channel with advantage not exceeding the sum of the advantages in the three intermediate steps.

The final thing to address is forward secrecy. We capture this by appealing to the memory cell we defined in the preliminaries (this is roughly the technique of [Jos20]), which provides an abstract means for an adversary to obtain a static private key. We will eventually arrive at a resource which is realised provided the memory cell is not corrupted *before a certain point*, yielding a constructive notion of forward secrecy. Our general technique for proving that we can construct these resources, even with corruption interfaces, is to programme the simulator to cause the ideal resource to behave the same as the real system in case of corruption. We can then reduce to a case with no corruption, whence the techniques are standard.

From Authentic Public Keys to Unilateral Keys We first describe a system that can be simulated by the Implicitly Unilateral n-Key Resource (recall this is abbreviated to ${}^{*}\mathsf{UKR}^{n}_{\mathsf{impl}}$). Consider a pair of channels: a one-time authenticated channel from the server to a client, and an insecure channel from that client to a server. Take the parallel composition of these resources, which we call a 'channel pair'. The assumed resources are then a parallel composition of

- A memory cell, which has n sub-interfaces
- n channel pairs

Let $\Psi = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ be a KEM. On the client side, the converters encapsulate against the server's public key (which they are assumed to receive authentically) and output the key they generate. On the server side, the converter decapsulates the ciphertexts it receives and outputs the corresponding keys. A formal definition of the converters can be found in Figure 6.

We construct ${}^{*}\mathsf{UKR}^{n}_{\mathsf{impl}}$ as follows. At the client side, to each channel pair attach sub-interface `collect-pk` of $\pi^{\mathsf{client}}_{\mathsf{impl}}$ to the authenticated channel; attach sub-interface `submit-ctxt` of $\pi^{\mathsf{client}}_{\mathsf{impl}}$ to the insecure channel. At the server-side, attach sub-interface `mem-cell` of $\pi^{\mathsf{server}}_{\mathsf{impl}}$ to the memory cell interface (we assume the memory cell is local to the server); attach the sub-sub-interfaces $\texttt{submit-pk}_i$ of $\pi^{\mathsf{server}}_{\mathsf{impl}}$ to the authenticated channel of one of the n channel pairs; and attach sub-sub-interface $\texttt{collect-ctxt}_i$ of $\pi^{\mathsf{server}}_{\mathsf{impl}}$ to the insecure channel running from the clients to the server.

We have the following.

Lemma 1. *The real system we have described $\epsilon(\kappa)$-realises ${}^{*}\mathsf{UKR}^{n}_{\mathsf{impl}}$ in the presence of a computationally bounded distinguisher. In particular, for any such dis-*

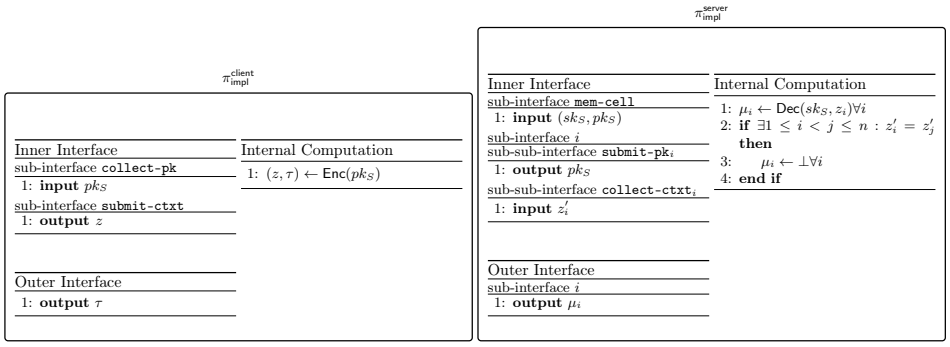


Fig. 6: The client-side (left) and server-side (right) converters impl, which construct implicitly unilateral keys from authentic copies of a public key. The server splits its inner interface into $n + 1$ sub-interfaces; one for locally requesting the static public key and decryption services from the memory cell, and n to keep track of the n parallel sessions it participates in. Note that we can trivially prevent adversarial key duplication by making the server abort if it receives two copies of the same ciphertext.

tinguisher there is a computationally bounded adversary $\mathcal{A}$ such that $\epsilon(\kappa) \leq n\mathsf{Adv}^{\mathsf{ind-cca}}_{\Psi,\mathcal{A}}(\kappa)$

Proof. The view of the distinguisher for the real system as follows: on the client side there are n sub-interfaces, which output secret keys, and similarly on the server side. At the adversarial interface, the distinguisher can collect n ciphertexts (and a copy of the static public key), and is allowed to submit n ciphertexts of its choice, such that the underlying keys will appear at server sub-interfaces of its choice. It also has access to a corruption interface; if activated, the distinguisher sees the secret key corresponding to the public key it saw.

With this in mind, the simulator works as follows. It generates a local key pair $(pk_{\mathsf{Sim}}, sk_{\mathsf{Sim}}) \leftarrow \mathsf{KeyGen}$ and generates n encapsulations $(z_i, \xi_i) \leftarrow \mathsf{Enc}(pk_{\mathsf{Sim}})$. Each z_i, and pk_{Sim}, is output to the adversary. When the simulator receives n ciphertexts z'_i, and the distinguisher's choice of whether or not to corrupt, at the adversarial interface, it prepares its input to the ideal resource as follows.

- If the adversary has not submitted a corrupt query, set $s = 0$. For each i:
  - If $z'_i = z_j$ for some $j \in [n]$, set $\sigma(j) = n$
  - If $z'_i \neq z_j$ for any $j \in [n]$, set $r_i \leftarrow 1$, $\nu_i \leftarrow \mathsf{Dec}(sk_{\mathsf{Sim}}, z'_i)$

  Suppose there are k indices for which r_i is set to 1; there remain k indices on which σ is not yet defined. There are also k' indices who do not have an inverse under σ; assign the latter to the former under σ bijectively. If a bijection is not possible to select, choose arbitrarily.
- If the adversary has submitted a corrupt query, set $s = 1$ and return sk_{Sim} to the distinguisher. For each i:

- Set $\bar{\mu}_i \leftarrow \xi_i$
- If $z'_i = z_j$ for any $j \in [n]$, set $\sigma(j) = n$, $r_i \leftarrow 1$, and $\nu_i \leftarrow \xi_\sigma^{-1}(i)$
- If $z'_i \neq z_j$ for any $j \in [n]$, set $r_i \leftarrow 1$ and $\nu_i \leftarrow \mathsf{Dec}(sk_{\mathsf{Sim}}, z'_i)$.

Reconstruct the remainder of σ in the same fashion as the previous step.

If a corrupt query is issued both systems are distributed identically; the distinguisher sees internally selected keys generated uniformly at random at the client interface, and ciphertexts bound to those keys with a known key pair. If it re-routes any ciphertexts, the same keys will appear at the server-side in either case (since the simulator uses the inject functionality to enforce the match), or if it injects distinct ciphertexts, decapsulations under the appropriate private key are output.

Suppose that a corrupt query is never issued. We proceed by means of a reduction: consider the reduction acting as an interface between the $\mathsf{n}-\mathsf{ind}-\mathsf{cca}$ security game and a distinguisher. When the game outputs challenge plaintexts $\mu_{b,i}$ these are output at each C interface; challenge ciphertexts and the public key are output at the adversarial interface. When the distinguisher submits its choice of ciphertexts at the adversarial interface, the reduction queries these ciphertexts to the game's decapsulation oracle, and outputs the plaintexts it receives at each server sub-interface.

If the game set its random choice of b as 0, the view of the distinguisher is precisely as though it interacts with the real system. If b was set to 1 the view of the distinguisher is precisely as though it interacts with the ideal system, because the simulator is generating its ciphertexts with respect to internally selected values independently of the $\mu_{b,i}$ output at the client interfaces. An adversary that simply runs this distinguisher as a subroutine and returns the b' it obtains wins the $\mathsf{n}-\mathsf{ind}-\mathsf{cca}$ security game with exactly the success probability of the distinguisher.

Finally, note that both systems abort at their client sides if and only if two equal ciphertexts are ever delivered at the adversarial interface.

From Unilateral Keys to Authenticated Channels We are now going to apply parallel instances of symmetric-key authentication protocols to the interfaces of the resource we have constructed, which in turn will construct the Parallel Authenticated Channels with Rewiring resource we have already seen. Because of the permutations we have to handle, the proof is a matter of rather fiddly bookkeeping.

Let $\mathcal{H} : \mathcal{M} \to \mathcal{T}$ be a family of strongly universal hash functions, and consider the parallel composition of the Implicitly Unilateral n-Key Resource and n insecure channels from the server to the clients. On the server side, the converters receive keys from ${}^*\mathsf{UKR}^n_{\mathsf{impl}}$ and message inputs at their outside interface, and send the resulting message-tag pairs over the insecure channels. On the client side, the converters receive a key from ${}^*\mathsf{UKR}^n_{\mathsf{impl}}$ and a message-tag input from an insecure channel, and verify whether the pair they have received is valid with respect to their key. A formal definition of the converters can be found in Figure 7.

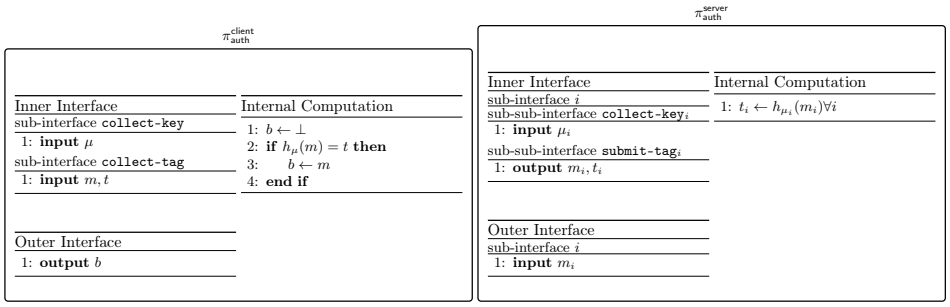


Fig. 7: The client-side (left) and server-side (right) converters auth, which construct parallel authenticated channels from implicitly unilateral keys.

We construct $\mathsf{AC}^n_\sim$ as follows. At the client side, to each client interface of ${}^*\mathsf{UKR}^n_{\mathsf{impl}}$ attach sub-interface `collect-key`; to each insecure channel attach sub-interface `collect-tag`. At the server side, to each server interface of ${}^*\mathsf{UKR}^n_{\mathsf{impl}}$ attach sub-sub-interface `collect-key`$_i$; to each insecure channel attach sub-sub-interface `submit-tag`$_i$.

We have the following.

Lemma 2. *The real system described above $n/|\mathcal{T}|$-realises ${}^*\mathsf{AC}^n$.*

Proof. A distinguisher's view of the real system is as follows: it can make n message inputs to the system on the server-side, and has access to the adversarial controls of the combined unilateral n-key resource, and the n insecure channels – so it can submit a permutation σ, and n pairs $(r_i, \bar{\tau}_i)$, to the unilateral n-key resource. It receives n honestly-generated (according to its selection of key distribution, including re-routing and injection) message-tag pairs, and can submit its own choice of n message-tag pairs and collect the corresponding message outputs at the client interfaces. Finally, it has access to a corruption interface, whereby once a particular flag is set to true it can input its own choice of secret keys, to be used by the client converters for verification.

Consider a version of ${}^*\mathsf{AC}^n$ in which corruption is global; that is, there is a single corruption command, after which all the authenticated channels accept messages of the distinguisher's choice. We show that this variant is realised by our construction.

We first have to define a simulator for this version of ${}^*\mathsf{AC}^n$. To avoid notational overlap, in the following let r, s be the injection and corruption flags for ${}^*\mathsf{UKR}^n_{\mathsf{impl}}$, and $\{p_i\}_i, q$ be the denial-of-service and corruption flags for ${}^*\mathsf{AC}^n$.

– Start by generating n internal secrets $\nu_1, ..., \nu_n$. When messages $m_1, ..., m_n$ are input at server sub-interfaces 1 to n respectively and input to the simulator at its inside interface, it generates $t_i \leftarrow h_{\mu_{\sigma^{-1}(i)}(m_i)}$ if $r_i = 0$, and $t_i \leftarrow h_{\mu'_i}(m_i)$ otherwise. The n pairs (m_i, t_i) are output at the adversar-

ial interface; suppose n pairs (m'_i, t'_i) are now received at the adversarial interface.

- If the corruption flag s is set to 1, set q to be 1. The corruption command comes with a vector of substitute client keys $(\bar{\mu}_1, ..., \bar{\mu}_n)$. For each i, check if $h_{\bar{\mu}_i}(m_{\sigma(i)}) = t_{\sigma(i)}$; if it does, set $\bar{m}_i \leftarrow m_{\sigma(i)}$, otherwise set $\bar{m}_i \leftarrow \perp$.
- If the corruption flag was not set to 0, for pair (m'_i, t'_i) proceed as follows.
  - If $r_i = 1$, set $p_i \leftarrow 1$ regardless of the message-tag validity.
  - If $r_i = 0$, check if $(m'_i, t'_i) = (m_{\sigma(i)}, h_{\nu_i}(m_{\sigma(i)}))$. If it does set $p_i \leftarrow 0$, otherwise set $p_i \leftarrow 1$.

The first thing to note is that if the corruption flag input at the adversarial interface is set to 1, both systems are distributed identically – if the message-tag pairs that were input at the adversarial interface verify with respect to the appropriate $\bar{\mu}$ as chosen by the distinguisher, those messages will be output at the proper client interface. Meanwhile, all message-tag pairs are generated according to the same distribution; tags are either computed with respect to a private key sampled uniformly, or a key of the distinguisher's choice.

It is known[5] that the distinguishing advantage between two systems is the sum over all input-output tuples of the difference between their joint probability distributions. For the tuples that the corruption flag is 1, since the joint probability distributions are identical, this difference is 0, so it remains to compute this same sum across all tuples where the corrupt flag is set to 0. But this is the same as computing the distinguishing probability between the two systems, each without a corruption functionality – and we can do this via a reduction as follows.

The following terminology will be useful in the rest of the argument: we will henceforth refer to those clients for which $r_i = 0$ as 'bound', otherwise clients are 'unbound'. We need to show that an arbitrary choice of s by the simulator yields a system indistinguishable from the real one. In this direction, we proceed by a reduction.

Let $\mathcal{H}$ be a family of strongly universal hash functions, and consider the following game. n keys $\{\mu_i\}$ are generated, and the adversary selects some subset of $\{1, ..., n\}$. Say this subset is $\{i_j\}$ for $1 \leq j \leq k$, the adversary has access to k single-use oracles that return $h_{\mu_{i_j}}(m_{i_j})$ for the adversary's choice of k messages, m_{i_j}. Eventually the adversary submits n message tag pairs (m_i, t_i). If at least one pair (m_i, t_i) verifies with respect to μ_i, and m_i was not queried to the μ_i oracle, the adversary wins.

Consider the reduction that sits between an adversary in this game and the candidate system. The adversary first has to choose which 'mode' to put the game into, submitting the (not necessarily proper) subset of $\{1, ..., n\}$ for which it wishes to activate single-message oracles. Notice that the bijection from bound clients to sub-interfaces can be chosen arbitrarily by the reduction (and the simulator may further choose a full bijection based on this partial bijection); the reduction does so. For unbound clients the reduction again picks the injected

[5] See a derivation of this, for example, in [Por14].

values $\bar{\mu_i}$ arbitrarily. These values are all submitted at the adversarial interface. When the adversary submits its queries m_{i_j}, the reduction inputs these at the appropriate (according to the partial bijection it selected internally) server sub-interfaces, and collects the resulting message-tag pairs; these are oracle outputs of the adversary's queries. When the adversary submits its n challenge pairs (m_i^*, t_i^*), the reduction inputs them on the insecure channels; the output at the client interfaces determines whether the game is won or not.

We have to verify that the two systems behave the same if and only if the distinguisher's transcript defines a losing event in the above game, and then compute the probability of winning that game. We can then apply Lemma 2 of [Mau13] to conclude that the distinguishing advantage is bounded by this probability[6].

Suppose the real and ideal systems behave differently. Since by definition, the message-tag pairs the system outputs are distributed identically in either case, the systems can only behave differently by the distributions of the outputs at the client interfaces being different. Suppose there is such a client, say client i; we split into the bound and unbound case. If the client is bound then we must have that $m_i^* \neq m_{s(i)}$, because the real and ideal systems are distributed identically if the distinguisher faithfully delivers messages according the permutation. But if indeed $m_i^* \neq m_{s(i)}$ then the ideal system always rejects, so the two can only behave differently if at some point a valid message-tag pair, with respect to a genuine client key, is produced, having not seen an evaluation of that message under that key previously. This constitutes a winning event. If the client is unbound the ideal system always rejects, and because the client's secret was never delivered no messages under that secret have been produced, so the systems can only behave differently if the game is won. The contrapositive then yields the reverse direction of the equivalence to be demonstrated.

In the other direction, suppose the systems are distributed identically. Since all other interface outputs are distributed identically by definition we only need to study the client outputs. Again, we analyse the bound and unbound cases separately. If the unbound client interfaces are all distributed identically then in the real case, all the client converters rejected, and so certainly the game is not won with respect to those message-tag pairs. If the bound client interfaces are distributed the same, either the correct message with respect to the permutation was delivered to that client, which means even a successful forgery does not define a winning event because a tag of that message under the relevant key has been seen; or a different message was delivered. In this case the ideal system always rejects, so if the systems behave identically the real system must also reject and the game is not won.

It remains to compute the probability of winning the game we have set up. Notice that since the n keys are selected independently and uniformly at random, the probability that a message-tag pair is valid with respect to a given key is independent of the other keys. For bound clients, the probability that a distinct

[6] This technique can be seen as a composable version of the usual identical-until-bad argument.

message-tag pair is valid under the relevant key is known to be $1/|\mathcal{T}|$ – this is as a consequence of the pairwise independence (i.e. Point 2 of Definition 1) of the hash functions, and is shown in [WC81] (and with a slightly more modern treatment in [Sti94]). In the unbound case, the key is a random value on which no information has been revealed, so the probability of winning is again $1/|\mathcal{T}|$ by Point 1 of Definition 1. All of the possible accept events are independent, so the total winning probability is bounded by $n/|\mathcal{T}|$.

Finally, we need to show the construction of this variant of ${}^*\mathsf{AC}^n$ suffices for the claim of the theorem. Call the variant "global" ${}^*\mathsf{AC}^n$; we argue that global ${}^*\mathsf{AC}^n$ can simulate ${}^*\mathsf{AC}^n$. Suppose a simulator receives n local corruption commands. If they are all 0 set the global corruption flag to 0; otherwise, set it to 1. For the local corruption flags that were set, input the distinguisher's choice of message; for the ones that were not, input the messages that were input on the server side. By composing simulators we are done.

From Authenticated Channels to Explicitly Unilateral Keys Now that we have successfully separated our sessions, we have at last returned to a regime where we can (more or less) argue independently about resources in the $\{A, B, E\}$ setting, and then take their parallel composition to obtain results in the single-server-many-client setting. In this direction we recall the method set out by [MTC13]: clients send ephemeral public keys via insecure channels to the server. The server can then encapsulate against this public key and return a ciphertext down the authenticated channel; however, they must also reply with a copy of the public key the received via the insecure channel, since otherwise the adversary has a misbinding attack[7]. In fact, it suffices for the server to reply with a *hash* of the public key, which will grant us a considerable improvement in bandwidth. By using cryptographic hash functions we will construct ${}^*(\mathsf{AC}^+)$ from the authenticated channels, the details of which are deferred to an appendix.

From Server-to-Client Authenticated Channels to Weak Client-to-Server Authenticated Channels.

From Weak Two-way Authenticated Channels To Explicitly Unilateral Keys Let Ψ be an $\mathsf{ind-cpa\ KEM}$, and consider a channel $({}^*\mathsf{AC}^+)^{C_i \leftarrow S_j}$. On the client side, the converter locally generates an ephemeral key pair (pk_e, sk_e), and sends pk_e via the channel to the server. It receives some message data at its inner interface from the channel (possibly taking the value $\perp$) and a confirmation value $\perp/\top$; if either of these values take $\perp$, output $\perp$ at the outer interface. Otherwise, output the decapsulation (under the local secret key) of the message data received. On the server side, the converter receives some message data at its inner interface. It encapsulates with respect to this data, outputting the key

[7] If the adversary delivers its own public key to the server it can read the key that the server selects by examining the resulting ciphertext. It can then compute its own encapsulation against the client's public key, thereby obtaining knowledge of the keys output at both honest party interfaces.

at its outer interface, and the ciphertext at another sub-interface of its inner interface. A formal definition of the converters can be found in Figure 8.

We construct ${}^*\mathsf{UKR}_{\mathsf{expl}}^{C_i \leftarrow S_j}$ as follows. At the client side, attach $\pi_{\mathsf{expl}}^{\mathsf{client}}$ to $(*\mathsf{AC}^+)^{C_i \leftarrow S_j}$ as follows: converter sub-interface `submit-pk` is attached to resource sub-interface `msg-in`; converter sub-interface `collct-traffic` is attached to resource sub-interface `msg-out`; and converter sub-interface `confirm` is attached to sub-interface `confirm`. At the server side, attach $\pi_{\mathsf{expl}}^{\mathsf{server}}$ to $(*\mathsf{AC}^+)^{C_i \leftarrow S_j}$ as follows: converter `collect-pk` is attached to resource sub-interface `msg-out`; and converter sub-interface `submit-traffic` is attached to resource sub-interface `msg-in`. We have the following.

Lemma 3. *The system we have described above $\epsilon(\kappa)$-realises ${}^*\mathsf{UKR}_{\mathsf{expl}}^n$ in the presence of a computationally bounded distinguisher. In particular, for any such distinguisher there is a computationally bounded adversary $\mathcal{A}$ such that $\epsilon(\kappa) \leq n\mathsf{Adv}_{\Psi,\mathcal{A}}^{\mathsf{ind-cpa}}$.*

Proof. The proof is similar to that of [MTC13, Lemma 4], except we have to account for the case where corruption occurs. As usual, this will be accomplished by our generic technique of programming the simulator to collapse to the real system in the case of corruption.

The view of the distinguisher for the real system is as follows: at the client and server sides there is a key output. At the adversarial interface the distinguisher sees a copy of an honestly generated public key pk_e, and an honestly generated ciphertext z. It can input a substitute public key pk'_e and a substitute ciphertext $\bar{z}$; as well as 3 control variables that govern how these values are delivered. To avoid notational overlap, in the following let s_1, s_2 be the control variables of ${}^*\mathsf{UKR}_{\mathsf{expl}}^{C_i \leftarrow S_j}$.

The simulator is built as follows.

- Start by generating (pk_e, sk_e) internally, and encapsulate (z, ν) against pk_e. When the simulator receives a public key pk'_e and a ciphertext $\bar{z}$ at the outer interface, encapsulate against pk'_e and set μ' to take the value of the resulting key; set $\bar{\mu} \leftarrow \mathsf{Dec}(sk_e, \bar{z})$.
- Set the values of (s_1, s_2) depending on the submitted values of (r_1, r_2, r_3) according to the following table (note that the value of r_3 is unimportant unless $r_1 = 2$).

(r_1, r_2, r_3)	(s_1, s_2)
$(0, 0, \cdot)$	$(0, 0)$
$(0, 1, \cdot)$	$(0, 1)$
$(1, 0, \cdot)$	$(1, 0)$
$(1, 1, \cdot)$	$(1, 1)$
$(2, 0, 0)$	$(2, 0)$
$(2, 0, 1)$	$(1, 0)$
$(2, 1, 0)$	$(2, 1)$
$(2, 1, 1)$	$(0, 1)$

Clearly the two systems behave the same when $r_1 = 2$. The remainder follows the argument of [MTC13, Lemma 4]; if $r_1 \neq 2$ then the two systems behave the same if $r_2 = 1$. In the remaining cases a standard reduction to the $\mathsf{KEM}\ \mathsf{ind}-\mathsf{cpa}$ game follows; the claim of the theorem follows from parallel composition.

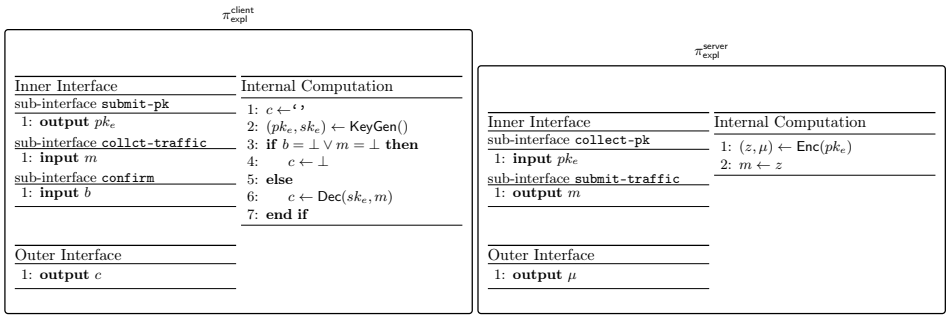


Fig. 8: The client-side (left) and server-side (right) converters expl, which construct explicitly unilateral keys from authenticated channels. Note that m can implicitly take the value $\bot$.

Theorem 2. *The converters*

$$\pi^{\mathsf{client}}_{\mathsf{expl}} \circ \pi^{\mathsf{client}}_{\mathsf{plus}} \circ \pi^{\mathsf{client}}_{\mathsf{auth}} \circ \pi^{\mathsf{client}}_{\mathsf{impl}} \quad \textit{and} \quad \pi^{\mathsf{server}}_{\mathsf{expl}} \circ \pi^{\mathsf{server}}_{\mathsf{plus}} \circ \pi^{\mathsf{server}}_{\mathsf{auth}} \circ \pi^{\mathsf{server}}_{\mathsf{impl}}$$

$\epsilon(\kappa)$*-construct* ${}^{*}\mathsf{UKR}^{n}_{\mathsf{expl}}$ *from* n *channel pairs and a memory cell in the presence of a computationally bounded distinguisher. In particular, let* Ψ *be the* KEM *used by the* impl *converters, let* $\mathcal{H} : \mathcal{M} \to \mathcal{T}$ *be the family of universal hash functions used by the* auth *converters, let* H *be a cryptographic hash function, and* Φ *be the* KEM *used by the* expl *converters; then for any such distinguisher there exist computationally bounded adversaries* $\mathcal{A}$*,* $\mathcal{B}$ *and* $\mathcal{C}$ *such that*

$$\epsilon(\kappa) \leq n \cdot \left(\mathsf{Adv}^{\mathsf{ind}-\mathsf{cca}}_{\Psi,\mathcal{A}}(\kappa) + \frac{1}{|\mathcal{T}|} + \mathsf{Adv}^{\mathsf{spr}}_{H,\mathcal{C}}(\kappa) + \mathsf{Adv}^{\mathsf{ind}-\mathsf{cpa}}_{\Phi,\mathcal{C}}(\kappa) \right)$$

Proof. Apply Lemmas 1, 2, 4 and 3, and the sequential composition theorem.

Remark 1. Since $\mathsf{ind}-\mathsf{cpa}$ security is a strictly weaker notion than $\mathsf{ind}-\mathsf{cca}$ security, one could use a single KEM twice, and still have an equivalent security guarantee. This will be appealing from the perspective of evaluating the NIST standard KEMs, which *only* come with $\mathsf{ind}-\mathsf{cca}$ guarantees.

Forward Secrecy We argue that the resources we have constructed natively model forward secrecy. Notice that the adversarial controls of ${}^{*}\mathsf{UKR}^{n}_{\mathsf{impl}}$ include

the control bits r_i, which correspond to whether the adversary faithfully delivers ciphertexts; and a global corruption bit s, which corresponds to learning the server's secret key. In particular, one can still trigger this corruption bit s after values of r_i have been selected and submitted; even after the protocol is complete, learning the server's secret key is equivalent to selecting the key output by the client (as we have seen), and there is no notion of forward secrecy. In the case of the corruptible authenticated channels we construct, however, the only notion of corruption we have is in specifying the behaviour of the resource; once this behaviour is specified there is, unlike in the ${}^*\mathsf{UKR}^n_{\mathsf{impl}}$ case, no post-hoc notion of corruption. This corresponds in the real world to the (almost tautological) fact that the key securing an authenticated channel can be leaked once that authenticated channel has undergone its single use. The resources we build on top of the authenticated channels are then forward secret in the sense that the notion of corruption is only valid when specifying the behaviour of the channel; if this behaviour is specified as non-corrupting, we do not get a later option to learn internal secrets. In particular, one can violate the security guarantees of the explicitly unilateral key resource only if the corruption is specified; this corresponds in the real world to the key established over the authenticated channel being independent of the key used to secure the authenticated channel.

4 Evaluation

In this section we compare the total size of the cryptographic objects transmitted in two protocols related to the spirit of our own, and CKAKE, with respect to some of the NIST standards. The two comparison protocols are KEM-TLS [SSW20], and the protocol implied by [MTC13], which we call MTC-13 for these purposes. We further use our trick of replacing the copy of the public key the server sends in MTC-13 with a hash of that public key – the security proof we have developed in this paper also applies to that setting.

Standards. The decision of which post-quantum schemes to include in our analysis is, to some extent, a matter of taste. Despite the establishment of official standards, spirited discussion on which alternative schemes are suitable for deployment continues within the community. We aim to keep as close as possible to the NIST standard recommendations while offering a diversity of options, and with this in mind consider:

- ML-KEM [ST24b]: the first-released and most efficient KEM standard, based on hard lattice problems
- ML-DSA [ST24a]: the first-released and fastest signature standard, with rather long signatures and public keys
- SLH-DSA [ST24c]: the third of the first round of NIST standards, based on security of cryptographic hash functions. It has very short public keys and very long signatures
- FN-DSA [Fou+18]: an alternative lattice-based signature, which is expected to be standardised imminently. It is competitive with ML-KEM in terms of

its space efficiency, but is slower than ML-DSA. Complications in its implementation arise from the need for floating-point arithmetic, and concerns within the community persist on this topic
- HQC [Mel+]: an alternative KEM slated for upcoming standardisation, which is much less efficient than ML-KEM but based on hard problems in linear coding theory, rather than lattices, and thus regarded as a necessary back-up option

There are various signature schemes with short signatures in development as part of NIST's signature 'on-ramp' programme, but these are considered far enough from standardisation to be out of scope for this analysis.

We are interested in the lengths of the public keys, and ciphertexts or signatures, since these are the objects that are transferred on the wire. Table 1 presents the length of these objects at the three NIST security levels.

	ML-KEM		**HQC**		**ML-DSA**		**SLH-DSA**		**FN-DSA**	
	pk	*ctxt*	*pk*	*ctxt*	*pk*	*sig*	*pk*	*sig*	*pk*	*sig*
1	800	768	2241	4433	1312	2420	32	7856	897	666
2	1184	1088	4514	8978	1952	3309	48	16224	-	-
3	1568	1568	7237	14421	2592	4627	64	29792	1793	1280

Table 1: The sizes of pertinent cryptographic objects across the post-quantum schemes we consider, in bytes, and considered at 3 levels of security.

Methodology. In each of the three protocols the client has to receive an authentic copy of the server's public key at some point. In practice this is usually accomplished by the transmission of a certificate (or possibly several intermediate certificates), containing the server's public key, various elements of metadata, and a signature[8] attesting to the validity of this data. From our perspective the only variable here is the size of the public key the server uses in the protocol, and so we only consider the bandwidth cost of the public key contained in the certificate.

The cryptographic objects transmitted on the wire by each protocol is detailed in Table 2.

We evaluate with respect to AES-128-GCM for the AEAD encryption, and SHA-256 for the cryptographic hash function (this is relevant both for computing HMAC tags and straight digests computed in the composable protocols). AES-128-GCM data is the size of the plaintext being encrypted, plus a 16-byte authentication tag; SHA-256 produces 32-byte digests. We also need to specify a universal hash function to compute our tags – in fact AES-128-GCM uses GHASH, a universal hash over $GF(2^{128})$, so this is a natural choice for our analysis. The tags are 16 bytes long.

[8] A signature is required for authenticity here – KEM-based authentication does not enjoy the property of public verifiability

KEM-TLS		MTC-13		CKAKE	
Client	*Server*	*Client*	*Server*	*Client*	*Server*
KEM public key	Ciphertext	16-byte nonce	Ciphertext	KEM public key	KEM public key
16-byte nonce	16-byte nonce	KEM public key	Signature public key	Ciphertext	Ciphertext
AEAD-encrypted ciphertext	AEAD-encrypted HMAC tag		Signature		Hash of KEM public key
AEAD-encrypted HMAC tag	AEAD-encrypted KEM public key		Hash of KEM public key		UHF tag

Table 2: The cryptographic objects transmitted on the wire by each of the protocols we consider. Items marked in blue correspond to the ephemeral portion of the protocol, items in red correspond to authentication.

All 3 protocols require an ephemeral KEM exchange – we fix ML-KEM as a control variable here, and change the available authentication mechanisms (be they KEMs or signatures).

Results. In Table 3 below we summarise the total length of the cryptographic objects sent on the wire in each of the 3 protocols we consider at each of the NIST security levels. We see that CKAKE authenticated with ML-KEM performs marginally better than KEM-TLS with respect to authentication with both KEMs, basically because of how the symmetric authentication portion works. With respect to authentication by either of the digital signature standards in the MTC-13 protocols the bandwidth is significantly better, and marginally better than with the as yet unstandardised FN-DSA signature.

KEMTLS	1			2			3		
	client	*server*	*total*	*client*	*server*	*total*	*client*	*server*	*total*
HQC	5313	3089	8402	10242	5682	15924	16069	8885	24954
ML-KEM	1648	1648	3296	2352	2352	4704	3216	3216	6432

MTC-13	1			2			3		
	client	*server*	*total*	*client*	*server*	*total*	*client*	*server*	*total*
ML-DSA	1328	4532	5860	1968	6381	8349	2608	8819	11427
SLH-DSA	48	8688	8736	64	17392	17456	80	31456	31536
FN-DSA	913	2363	3276	-	-	-	1809	4673	6482

CKAKE	1			2			3		
	client	*server*	*total*	*client*	*server*	*total*	*client*	*server*	*total*
HQC	5233	3057	8290	10162	5650	15812	15989	8853	24842
ML-KEM	1568	1616	3184	2272	2320	4592	3136	3184	6320

Table 3: The bandwidth required for the cryptographic objects transmitted in the three protocols we consider, with a fixed ML-KEM handshake.

Round Trips. As seems to be endemic to KEM-authenticated protocols, CKAKE needs one-and-a-half round trips, but is server-initiated (unlike KEM-TLS). In KEM-TLS the client is able to start sending data encrypted under implicitly authenticated keys after one round trip, which is not possible for us. However, we have the property that if the server's public key is pre-distributed, we can start sending data under explicitly authenticated keys after one round trip.

5 Conclusion

We have described a KEM-authenticated AKE protocol with provably composable security, that is competitive with similar protocols in terms of bandwidth. Two main areas for further study remain.

First, we would like to have a parallel composition theorem that accounts for shared state. This would be extremely useful as a general purpose tool for single-server-many-client scenarios, and would greatly simplify the proof of moving between the implicitly unilateral keys and authenticated channels.

Second, it would be useful to have a standardised $\mathsf{ind}-\mathsf{cpa}$ KEM. Both our protocol and that of [MTC13] demonstrate that the KEM used for the ephemeral portion of the protocol can be $\mathsf{ind}-\mathsf{cpa}$ secure (in comparison to KEM-TLS, which requires its ephemeral KEM to tolerate a single decapsulation query); since the general method of obtaining an $\mathsf{ind}-\mathsf{cca}$ KEM is to apply an overhead-inducing transform to an $\mathsf{ind}-\mathsf{cpa}$ KEM, we could expect to make our protocol more efficient.

References

[BCK98] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. "A modular approach to the design and analysis of authentication and key exchange protocols". In: *Proceedings of the thirtieth annual ACM symposium on Theory of computing*. 1998, pp. 419–428.

[Boy+08] Colin Boyd et al. "Efficient one-round key exchange in the standard model". In: *Australasian Conference on Information Security and Privacy*. Springer. 2008, pp. 69–83.

[BR06] Mihir Bellare and Phillip Rogaway. "Code-based Game-Playing Proofs and the Security of Triple Encryption". In: *Advances in Cryptology — EUROCRYPT 2006*. Vol. 4004. Lecture Notes in Computer Science. Fundamental lemma of game-playing gives the bound for identical-until-bad games. Springer, 2006, pp. 1–20. URL: `https://eprint.iacr.org/2004/331.pdf`.

[CG10] Ran Canetti and Sebastian Gajek. "Universally composable symbolic analysis of diffie-hellman based key exchange". In: *Cryptology ePrint Archive* (2010).

[CMT13] Sandro Coretti, Ueli Maurer, and Björn Tackmann. "A constructive perspective on key encapsulation". In: *Number Theory and Cryptography: Papers in Honor of Johannes Buchmann on the Occasion of His 60th Birthday*. Springer, 2013, pp. 226–239.

[CW77] J Lawrence Carter and Mark N Wegman. "Universal classes of hash functions". In: *Proceedings of the ninth annual ACM symposium on Theory of computing*. 1977, pp. 106–112.

[FO99] Eiichiro Fujisaki and Tatsuaki Okamoto. "Secure integration of asymmetric and symmetric encryption schemes". In: *Annual international cryptology conference*. Springer. 1999, pp. 537–554.

[Fou+18] Pierre-Alain Fouque et al. "Falcon: Fast-Fourier lattice-based compact signatures over NTRU". In: *Submission to the NIST's post-quantum cryptography standardization process* 36.5 (2018), pp. 1–75.

[Fuj+15] Atsushi Fujioka et al. "Strongly secure authenticated key exchange from factoring, codes, and lattices". In: *Designs, Codes and Cryptography* 76.3 (2015), pp. 469–504.

[Hül+21] Andreas Hülsing et al. "Post-quantum wireguard". In: *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2021, pp. 304–321.

[Jos20] Daniel Jost. "On Generalizations of Composable Security". PhD thesis. ETH Zurich, 2020.

[Koh+15] Markulf Kohlweiss et al. "(De-) constructing TLS 1.3". In: *International Conference on Cryptology in India*. Springer. 2015, pp. 85–102.

[Kra96] Hugo Krawczyk. "SKEME: A versatile secure key exchange mechanism for internet". In: *Proceedings of internet society symposium on network and distributed systems security*. IEEE. 1996, pp. 114–127.

[KW16] Hugo Krawczyk and Hoeteck Wee. "The OPTLS protocol and TLS 1.3". In: *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE. 2016, pp. 81–96.

[Mau13] Ueli Maurer. "Conditional equivalence of random systems and indistinguishability proofs". In: *2013 IEEE International Symposium on Information Theory*. IEEE. 2013, pp. 3150–3154.

[Mel+] Carlos Aguilar Melchor et al. "Hamming quasi-cyclic (HQC)". In: ().

[MR11] Ueli Maurer and Renato Renner. "Abstract Cryptography". In: *Proceedings of the Second Symposium on Innovations in Computer Science (ICS 2011)*. Ed. by Bernard Chazelle. Beijing, China: Tsinghua University Press, Jan. 2011, pp. 1–21.

[MTC13] Ueli Maurer, Björn Tackmann, and Sandro Coretti. "Key exchange with unilateral authentication: Composable security definition and modular protocol design". In: *Cryptology ePrint Archive* (2013).

[Por14] Christopher Portmann. "Key recycling in authentication". In: *IEEE Transactions on Information Theory* 60.7 (2014), pp. 4383–4396.

[Res18] E. Rescorla. *The Transport Layer Security (TLS) Protocol Version 1.3*. RFC 8446, Internet Engineering Task Force. Standards Track. Aug. 2018. URL: `https://doi.org/10.17487/RFC8446`.

[SSW20] Peter Schwabe, Douglas Stebila, and Thom Wiggers. "Post-quantum TLS without handshake signatures". In: *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. 2020, pp. 1461–1480.

[SSW21] Peter Schwabe, Douglas Stebila, and Thom Wiggers. "More efficient post-quantum KEMTLS with pre-distributed public keys". In: *European Symposium on Research in Computer Security*. Springer. 2021, pp. 3–22.

[ST24a] National Institute of Standards and Technology. *Module-Lattice-Based Digital Signature Algorithm Standard (ML-DSA)*. FIPS 204, Federal Information Processing Standards. Digital signature algorithm standard based on CRYSTALS-Dilithium. Aug. 2024. URL: `https://csrc.nist.gov/pubs/fips/204/final`.

[ST24b] National Institute of Standards and Technology. *Module-Lattice-Based Key-Encapsulation Mechanism Standard (ML-KEM)*. FIPS 203, Federal Information Processing Standards. Key encapsulation mechanism standard based on CRYSTALS-Kyber. Aug. 2024. URL: `https://csrc.nist.gov/pubs/fips/203/final`.

[ST24c] National Institute of Standards and Technology. *Stateless Hash-Based Digital Signature Algorithm Standard (SLH-DSA)*. FIPS 205, Federal Information Processing Standards. Digital signature algorithm standard based on SPHINCS+. Aug. 2024. URL: `https://csrc.nist.gov/pubs/fips/205/final`.

[Sti94] Douglas R. Stinson. "Universal hashing and authentication codes". In: *Designs, Codes and Cryptography* 4.3 (1994), pp. 369–380.

[WC81] Mark N Wegman and J Lawrence Carter. "New hash functions and their use in authentication and set equality". In: *Journal of computer and system sciences* 22.3 (1981), pp. 265–279.

A Constructing Weak Two-way Authenticated Channels.

The basic idea of the method of [MTC13] is to convert a server-to-client authenticated channel into a weaker variant of a client-to-server authenticated channel, by having the server replay the value it received via an insecure channel down the authenticated channel. In other words, there is no guarantee that the message the server received originated with the client, but the client will be notified if its legitimate message was not delivered.

Intuitively, because the information contained in the reply from the server is never used directly, and only checked for equality with local state, it should suffice to reply with a hash of this received data. Indeed this is possible, at the cost of incurring a second pre-image resistance term in the final security bound. For the purposes of the security analysis it will be convenient to introduce an intermediary resource, which we can think of as upgrading the existing authenticated channels we have constructed to also possess this weaker authentication in the direction. We call such a resource $({}^{*}\mathsf{AC}^{n}_{\sim})^{+}$; because our sessions are now

separated we can argue about this resource in the $\{A, B, E\}$ setting. For a formal definition of $({}^*\mathsf{AC}^n_\sim)^+$ see Figure 2.

Let H be a cryptographic hash function, and consider the parallel composition of ${}^*\mathsf{AC}^n$, and n insecure channels from the clients to the server. On the server side, converters get a message m at an internal sub-interface, and a message m_{Serv} at an outer sub-interface. The converters computes a hash h of m, and submit m_{Serv}, h at an internal interface. On the client side, the converters submit messages they receive at their outer interface to the insecure channel. When they receive data from the authenticated channel they parse it into a message sent by the server, and a tag resulting from the hash of the message they sent down the insecure channel; successful receipt by the server of that message is given only if a hash of the converter's internal state matches the received tag. A formal definition of the converters can be found in Figure 9.

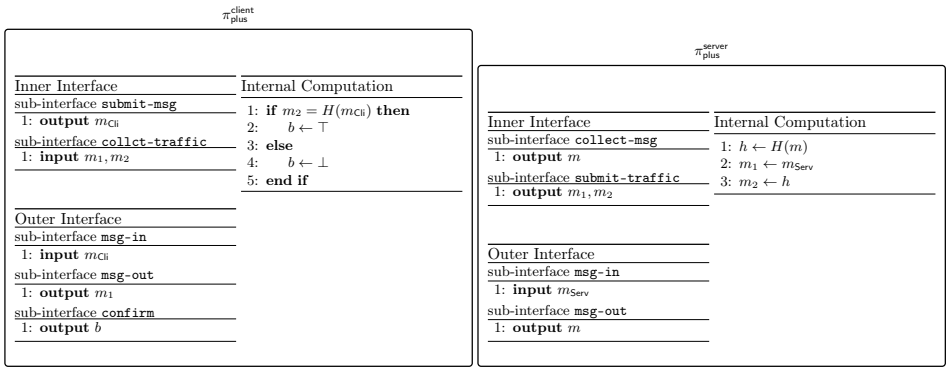


Fig. 9: The client-side (left) and server-side (right) converters plus, which construct explicitly unilateral keys from authenticated channels.

We construct $({}^*\mathsf{AC}^n_\sim)^+$ as follows. At the client side, to each client interface of ${}^*\mathsf{AC}^n$ attach sub-interface `collct-traffic`; to each insecure channel attach sub-interface `submit-msg`. At the server side, to each server sub-interface of ${}^*\mathsf{AC}^n$ attach sub-interface `submit-traffic`; to each insecure channel attach sub-interface `collect-msg`. We have the following.

Lemma 4. *The system we have described above $\epsilon(\kappa)$-realises $({}^*\mathsf{AC}^n_\sim)^+$ in the presence of a computationally bounded distinguisher. In particular, for any such distinguisher there is a computationally bounded adversary $\mathcal{A}$ such that $\epsilon(\kappa) \leq n\mathsf{Adv}^{\mathsf{spr}}_{H,\mathcal{A}}(\kappa)$.*

Proof. The view of the distinguisher for the real system is as follows: on the client side there is a message input, a message output, and a confirm/reject output. On the server side there is a message output, and a message input. At

the adversarial interface, the distinguisher has access to the controls of both the insecure channel and the corruptible authenticated channel; so it sees authentic copies of m_{Cli}, m_{Serv} and h, and can submit its own version of the client's message m'_{Cli} for the server to receive. If it further decides to corrupt the authenticated channel it can choose its own values $\bar{m}_{\mathsf{Serv}}$ and $\bar{h}$ to submit to the client.

The simulator is built as follows. It outputs the values $m_{\mathsf{Cli}}, m_{\mathsf{Serv}}$ it receives at its inner interface at its outer interface, and further outputs $h \leftarrow H(m_{\mathsf{Cli}})$. The simulator a

- If the authenticated channel control r is set to 2, set $r_1 \leftarrow 2$. If $m'_{\mathsf{Cli}} = m_{\mathsf{Cli}}$ set $r_2 \leftarrow 0$, otherwise $r_2 \leftarrow 1$; if $\bar{h} = H(m_{\mathsf{Cli}})$ set $r_3 \leftarrow 0$, otherwise $r_3 \leftarrow 1$.
- If the authenticated channel control r is set to 1, set $r_1 \leftarrow 1, r_2 \leftarrow 1$.
- If the authenticated channel control r is set to 0, set $r_1 \leftarrow 0$. If $m'_{\mathsf{Cli}} = m_{\mathsf{Cli}}$ set $r_2 \leftarrow 0$, otherwise $r_2 \leftarrow 1$.

If $r = 2$ the real system and ideal systems behave the same, so by the same argument we have seen before we can consider distinguishing advantage on non-corruptible versions of the resources. Again we proceed by a reduction; consider a reduction that sits between the system and the second pre-image resistance game with respect to the hash function H. The reduction picks a challenge value m_{Cli} and submits it at the client interface, as well as to the adversary. When the adversary submits its guess of a second pre-image, this is set as the value of m'_{Cli} and submitted at the adversarial interface. The reduction chooses the value of m_{serv} arbitrarily. The game is won if $m'_{\mathsf{Cli}} \neq m_{\mathsf{Cli}}$, and $\top$ is output at the `confirm` interface.

If $m'_{\mathsf{Cli}} = m_{\mathsf{Cli}}$ the two systems behave the same (recall that there is no corruption, so the value of h the server computes is always transmitted faithfully). Otherwise, the only way the two systems can behave differently is if the game-winning event occurs, so the distinguishing advantage is bounded by the adversary's advantage in a second pre-image resistance game as required. The result follows from parallel composition.