Generic Forward-Secure Key Agreement Without Signatures

Cyprien de Saint Guilhem, Nigel P. Smart, and Bogdan Warinschi

Dept. Computer Science, University of Bristol, United Kingdom.

Abstract. We present a generic, yet simple and efficient transformation to obtain a forward secure authenticated key exchange protocol from a two-move passively secure unauthenticated key agreement scheme (such as standard Diffie–Hellman or Frodo or NewHope). Our construction requires only an IND-CCA public key encryption scheme (such as RSA-OAEP or a method based on ring-LWE), and a message authentication code. Particularly relevant in the context of the state-of-the-art of postquantum secure primitives, we avoid the use of digital signature schemes: practical candidate post-quantum signature schemes are less accepted (and require more bandwidth) than candidate post-quantum public key encryption schemes. An additional feature of our proposal is that it helps avoid the bad practice of using long term keys certified for encryption to produce digital signatures. We prove the security of our transformation in the random oracle model.

1 Introduction

Forward secrecy and authentication are the standard security requirements for authenticated key agreement protocols (AKA). They require that parties authenticate one another, and that the key derived remains secret to anyone but to the two parties involved at the time of the execution. Modern realizations rely on the Diffie–Hellman protocol which is unauthenticated and guarantees key secrecy only against passive adversaries. The stronger property is obtained via additional mechanisms which authenticate the two parties and ensure integrity of the conversation between them, even against active adversaries.

Numerous generic transformations in the literature show how to achieve full AKA active security from protocols with weaker guarantees [3,9,22,28,18,24] using simple mechanisms such as signatures, encryption, and MACs.

Such generic techniques are particularly appealing; on the one hand they enable a modular approach where the base protocol and the details of the transformation are designed and analyzed independently – in particular, if needed, the underlying protocol can be easily swaped out and replaced with a different mechanism. On the other it provide conceptual clarity for choices that are made, e.g. which part of the the protocol provides say, key-secrecy, and which deals with integrity/entity authentication.

In this paper we contribute to this research direction. We provide a simple generic transformation which, when applied to a certain class of passively secure key-exchange protocols, yields the most round-efficient authenticated key-agreement protocols against

This work has been supported in part by ERC Advanced Grant ERC-2015-AdG-IMPaCT.

active adversaries to date. Besides optimal round complexity, our proposal has two interesting implications which serve as further motivation for this work. The first concerns the practicalities of existing RSA certified public keys; the second concerns security of key-agreement protocols in the post-quantum world.

Consider the instantiation of the "signed Diffie-Hellman" construction which appears, for example, in the popular TLS 1.0-1.2 ciphersuite

`TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256`,

using RSA signatures and elliptic curve based Diffie–Hellman. This usage is a bit of a kludge: RSA certificates in existence were issued for dual-use of RSA in both signature and encryption mode (which was needed for the earlier TLS mechanism of RSA key-transport which is still prevalent). Deploying protocols where the same keys are used for both signature and encryption would encourage a usage which is not supported by rigorous mathematical guarantees. Short of issuing new RSA keys, this type of misuse could be avoided by ensuring that existing keys are only used for encryption. We note that this is not just a theoretical concern. Attacks against deployed cryptography that reuse keys in unintended ways have been previously reported [27,19,20].

We now discuss the design of key-exchange protocols secure in the post-quantum setting. Here, a natural strategy is to consider existing designs and replace the different components with post-quantum secure versions. The underlying Diffie-Hellman constructions can be replaced by (Ring-)LWE-based variant such as NewHope [7] or Frodo [1]. For other primitives, the situation seems to be more delicate. Both for historical and technical reasons, there seems to be less confidence in proposals for post-quantum signatures th an for post-quantum encryption. Whilst lattice based encryption schemes have a strong track record, see NTRU [16] for a h istoric scheme or Ring-LWE [26,25] for more modern ones, the use of lattice based signature s chemes is less stable. Many early schemes, such as GGH [13] and NTRUSign [17,15], were eventually broken due to issues with the distribution of the signatures [12,29]; however recently more promising lattice based candidates have been proposed such as [10]. Post-quantum signature schemes based on Merkle hash trees have also had issues related to the need to maintain a large state; again recently this issue has been overcome with the introduction of state-less hash tree based [5].

Questionable dual use of RSA keys, and the relatively slow progress of post-quantum secure signature schemes, raises the question of whether one can design a passively (forward) secure unauthenticated protocol together with authentication mechanisms that rely solely on post-quantum public key encryption schemes.

Our results. We answer this question in the positive. We propose a generic transformation which bootstraps a forward secure AKA protocol out of a two-pass passively secure unauthenticated key agreement (KA) scheme which satisfies some mild additional conditions. The transformation uses an arbitrary IND-CCA public key encryption scheme and a strongly unforgeable MAC. Below we provide a sketch of our transformation, motivate its design and discuss the additional requirements on the underlying protocol.

Consider an arbitrary such protocol Π, whose execution between parties U and V is described in Figure 1 using the general syntax introduced by Bellare and Rogaway [4]. For example, Diffie–Hellman is an instantiation where U's ephemeral key is e_A and m_1 is g^{e_A}, V's ephemeral key is e_B (which can be deleted as soon as it is used to

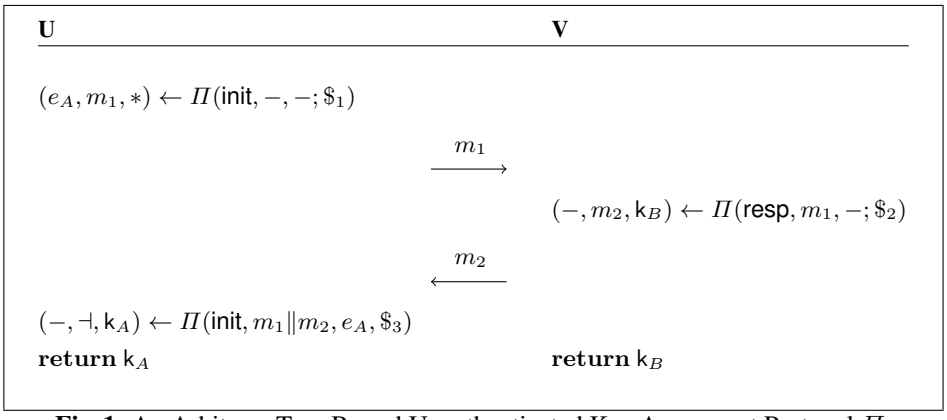

Fig. 1: An Arbitrary Two-Round Unauthenticated Key Agreement Protocol Π.

derive $m_2 = g^{e_B}$ and $\mathsf{k}_B = m_1^{e_B}$), finally the computation of k_A is done by U using the equation $\mathsf{k}_A = m_2^{e_A}$. To obtain forward secrecy, the ephemeral key data is assumed to be deleted as soon as the session keys are locally computed.

We bootstrap this two round KA protocol into a fully authenticated one (which inherits the forward secrecy property). Our construction, presented in Figure 2, requires a public key encryption scheme secure under chosen ciphertext attacks, a strongly unforgeable message authentication code, and two key derivation functions H_1 and H_2 which we model as random oracles.

The protocol works by wrapping the message flows, m_1 and m_2, of the KA protocol in encryptions under the long term keys of the two parties. Interestingly, the main role played by encryption here is to authenticate the parties and ensure integrity of the messages they exchange. Indeed, one can think of the first two messages of the protocol as a challenge-response exchange where U attempts to authenticate V by sending an encryption of m_1 under the public key of V and expecting to receive the same m_1 in the next flow. Similarly, the second and third flow can be interpreted as a challenge-response where V sends m_2 to U and expects to receive a message that depends on m_2. In addition, the MAC send as the last message also ties the identities of the parties involved with this particular execution of the protocol run. The final application key is derived from the same key from Π, but in a way that decouples it from the MAC key and also incorporates the identities of the participants.

The last message flow and key derivation methodology also thwart an analogue of the (in)famous attack against the Needham-Schroeder protocol. A malicious V could reencrypt the first message for a third party W who would reply with its own encrypted m_2 for U; V could simply forward this message so U. Parties U and W would thus derive the same key for the underlying passively secure protocol. However, W will no longer accept the MAC as it will be on the wrong message ($U\|V$ as opposed to $U\|W$), thus thwarting the attack. In addition, since it depends on the participants' identities, the derived session key will also be different for U and W.

The essence of our transformation is that it attempts to ensure that an active adversary cannot interfere with the execution of the underlying protocol, i.e. that when a

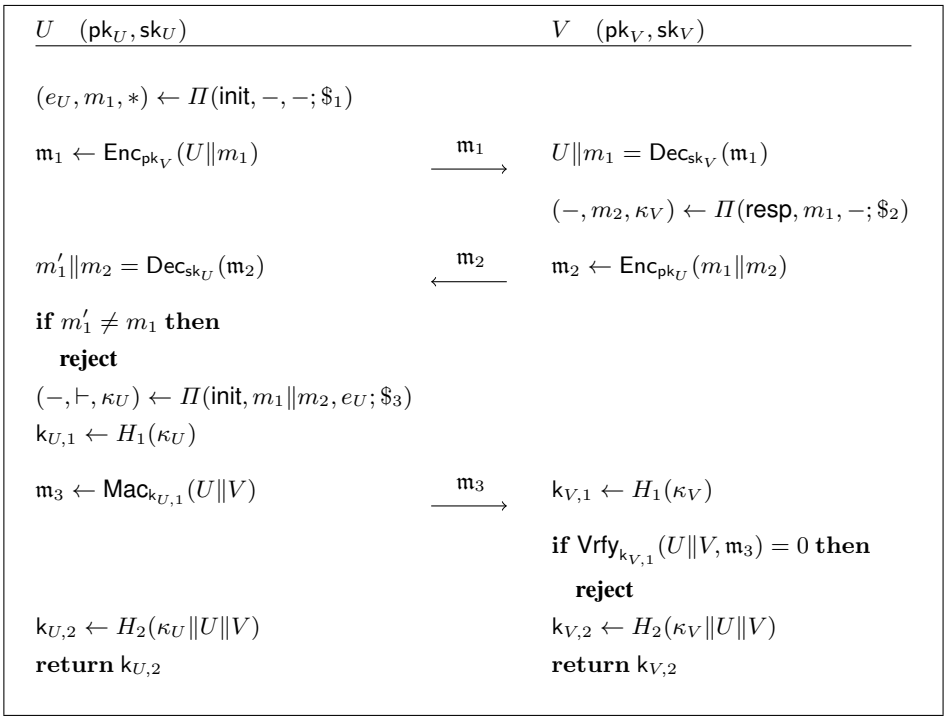

Fig. 2: The New AKA Protocol Construction.

party accepts, it must have engaged in an execution with another honest party. Put otherwise, even an active adversary cannot force a session to accept other than by forwarding honest messages.

Using non-malleable encryption to protect the integrity of messages goes someway towards implementing this intuition. Ensuring that parties authenticate each other successfully is however not obvious, and in fact require additional properties on the underlying protocol Π. As explained above, one should think of the first two messages as a challenge-response protocol to authenticate V. Notice that for security of authentication, this requires that message m_1 of the Π has sufficient entropy; otherwise, an adversary who guesses m_1 can reply with an appropriately message which encrypts m_1 and some m_2 and get U to accept.

Similarly, one should think of the second and third messages as a challenge-response protocol that authenticates U: the last message should only be computable by some party which received m_2 and derived the MAC key from it. This intuition is valid only if m_2 actually helps determine the MAC key, which is not necessarily the case. Consider a two message protocol where, if the first message of U for V is some fixed message bad, then V sets the local key to, say, 0^n. Such a protocol may still be secure against a passive adversary as an honest execution U would never send bad. Yet, the protocol obtained by applying our transformation is not actively secure since the adversary can send the encryption of bad to V. More generally, a close look shows that the problem

is that the adversary can send an appropriately crafted message m_1 which coerces the key into one which can be easily guessed (even if V behaves honestly).

The above discussion shows that we need two additional properties for our transformation to work: i) that the first message of Π is unpredictable and ii) that even if the first message is an arbitrary message sent by the adversary then the key derived by V is still unpredictable.

Naturally, one can ask if further subtle attacks are possible. We show that this is not the case and provide rigorous guarantees for the above intuition. We show that if the starting protocol is an arbitrary passively secure two-message protocol and satisfies the two additional security properties informally described above, then the transformation that we propose yields a full fledged forward secure key exchange protocol with mutual authentication (in the random oracle model), under standard assumptions on the encryption and MAC scheme used in the transformation.

Related work. The first generic compilers for authenticated key exchange were by Bellare, Canetti, and Krawczyk [3] later refined by Canetti and Krawczyk [9]. These works consider adversaries of different strength, but share an interesting idea of protocol design. First construct a protocol secure in a model where links between parties are authenticated (i.e. secure against passive adversaries), and then compile it into a stronger version, secure in a world with unauthenticated links, by using special-purpose *authenticators* which authenticate the sender of each message and ensure their integrity. In particular, BCK present an authenticator that uses IND-CCA2 secure encryption and MAC schemes. However, the use of authenticator replaces every message flow of the base protocols with three flows, so starting from a two-message flow protocols one obtains a stronger protocol that requires five rounds. Unfortunately the general setting of MT-authenticators of BCK works does not immediately allows for further optimisation which reduces the number of rounds.

Katz and Young[22] consider the problem of boosting passive security to active security for *group* key exchange by first exchanging nonces between parties and then authenticating each message through signatures that involve these nonces. For the case of two parties this result in a protocol with four message flows. For this type of protocols, a less efficient compiler is the one studied by Morrissey, Smart and Warinschi [28]. They show that TLScan be regarded as TLS as the successive applications of two generic transformations which bootstrap passive security to active security.

A second line of work which is related to ours is based on the observation that key encapsulation mechanisms naturally give rise to passively secure key-exchange protocols (where one party sends the parameters of a KEM scheme, and the second party sends a KEM). There are by now several constructions of key-exchange protocols (in settings which are sometimes different from ours) which start from KEMs. For example, Boyd et al. [8] construct authenticated key exchange from KEMs, meeting the eCK stronger security requirement, and Gunther et al [14] show how to add forward security to KEMs to obtain forward security when these are used as a full-key exchange protocol that enables forward secure 0-RTT. Both transformations work in the ID-based setting, use pairings and therefore are not generic.

Perhaps the closest work with ours is that of Li *et al*. [24] who present two transformations that bootstrap AKA protocols out of passively secure ones, one based on sig-

natures and another based on encryption. Both transformations first execute a passively secure KA protocol and then use three additional flows to perform entity authentication (and ensure the integrity of the conversation between the two parties). Just like our proposal, the encryption-based construction of [24] can serve to avoid the two issues which we have outlined above but at an increased round-complexity cost. In essence, we avoid additional communication rounds by showing how to piggy-back entity authentication on top of the passively secure protocol.

One observations which is warranted at this point is that our transformation does not achieve key-confirmation [11] (while derived keys are secret and parties authenticate each other, one party may accept without the other party actually having derived the key), whereas some other transformations do. This was not an explicit goal, afterall the notion has only recently been formalized [11].

2 Preliminaries

We first recall some standard definitions of primitives and their security notions. A comprehensive overview of this material can be found in [21] and in the full version of this paper. We then recall basic notions of security for passive key agreement protocols and introduce two new formal definitions. Throughout this paper, we denote the security parameter by λ, represented in unary notation as 1^λ, and the empty string by $-$.

2.1 Standard Definitions

We recall briefly the informal descriptions of actively secure public-key encryption schemes (with the addition of multi-user security), strongly unforgeable message authentication codes as well as key derivation functions and the random oracle model.

Public-key encryption schemes. In this paper, we denote a public-key encryption scheme by a tuple $E = (\mathsf{Setup}, \mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ of $\mathsf{poly}(\lambda)$-time algorithms. We assume that such schemes correctly decrypt honestly encrypted ciphertexts with overwhelming probability.

The standard (single-user) active security notion for such schemes is that of *indistinguishability under chosen ciphertext attack*, denoted IND-CCA. The experiment, also called game, for this setting gives an arbitrary adversary a randomly sampled public-key and, upon query of a left-right oracle, denoted L-R, with two messages of identical lengths, returns the encryption of one of the two. Given access to a decryption oracle, the adversary's goal is to guess which of the two messages the oracle encrypts. The adversary may query either oracle several times, with the only restriction that it may not query the decryption oracle on any ciphertext output by the left-right oracle.

In the proof of security of our protocol, we make use of the multi-user security notion described in [2]. For n participants, the n-IND-CCA security experiment is very similar to the single-user setting. The difference is that the adversary is provided with n different public keys and may query the left-right oracle on any one of these keys. Whether it is the right or left message which is encrypted is still selected at random, but this choice remains consistent between all queries of the L-R oracle.

Unforgeable message authentication codes. Message authentication codes (MACs) are symmetric key primitives that allow parties sharing a secret key k to authenticate and verify messages, thus detecting eventual modification of their content. A MAC is a triple of $\mathsf{poly}(\lambda)$-time algorithms $M = (\mathsf{KGen}, \mathsf{Mac}, \mathsf{Vrfy})$ such that, given a message and a key, Mac produces a tag, and such that, given a message, a tag and a key, Vrfy verifies that the tag corresponds to the message.

The security experiment for strong unforgeability, denoted $\mathsf{MAC\text{-}sFORGE}$, generates a random key and gives the adversary access to a Mac oracle whilst recording pairs of queried messages and the tag that was returned for each. The goal of the adversary is to output a message and a tag such that the verify algorithms accepts this tag and such that this tag was never produced by the Mac oracle for this message.

Key derivation functions and the random oracle model. In cryptographic schemes such as key agreement protocols, the secret information that is exchanged often cannot be used "out of the box" to achieve other goals such as encryption or authentication. Instead, we must use a method to transfer the high entropy of the key agreement session key into a format that is more suitable. This is acheived by making use of *key derivation functions* (KDFs) which are functions with high min-entropy, i.e. an adversary has a negligible chance of correctly guessing the output computed from a given input. While in practice great care must be given to the instanciation of such a KDF, we will make use here of the *random oracle model* and assume that the KDFs we use sample their output uniformly at random from a given space. We will use two independent random oracles which we will denote by H_1 and H_2.

2.2 Passively Secure Unauthenticated Key Agreement Protocol

First, we formalise what we mean by a (simple) unauthenticated key agreement protocol and what it means for such a protocol to be passively secure. Informally we consider a protocol passively secure if an adversary cannot determine the session key from seeing a transcript. We make no usage of long term keys at this stage, as we are focusing on unauthenticated protocols. In a later section we will discuss the model for fully actively secure, and authenticated, key agreement.

Informally, a key agreement protocol is a set of instructions, executed by two parties involved in a conversation, which leads to both of them computing identical session keys. These keys are then usually used to authenticate or encrypt further communication. The most basic security notion expected of such a protocol is that an adversary who has access to the transcript of a conversation is incapable of obtaining any information regarding the final session key. Our formalisation below is inspired by the original definition of such protocols by Bellare and Rogaway [4].

Definition 1 (Unauthenticated Key Agreement Protocol). *An* unauthenticated key agreement protocol *is a pair of probabilisitc* $\mathsf{poly}(\lambda)$*-time algorithms* (Setup, Π) *such that:*

1. *The* setup *algorithm* Setup *takes as input the security parameter* 1^λ *and outputs a tuple of public parameters,* params*, required by the key agreement protocol. Amongst other information* params *specifies a message space* $\mathcal{M}$ *and a key space* $\mathcal{K}$*. We assume for convenience that* λ *is implicit in* params*.*

2. *The* protocol function Π *is a function that dictates which messages the participating entities should compute and send to one another. Its input and output are of the form* $(\epsilon', m, \delta, \kappa) \leftarrow \Pi(\mathsf{params}, \rho, \tau, \epsilon; \$)$ *where the inputs are defined by:*
   - params *are the system parameters.*
   - $\rho \in \{\mathsf{init}, \mathsf{resp}\}$ *is the role of the entity running the function.*
   - $\tau \in \{0,1\}^*$ *is a transcript of the conversation so far.*
   - $\epsilon \in \{0,1\}^* \cup \{\perp\}$ *is ephemeral state information which needs to be passed from one party's invocation of* Π *to the next.*
   - $\$$ *is some randomness.*

   And the outputs of Π *are given by*
   - $\epsilon' \in \{0,1\}^* \cup \{\perp\}$ *is updated state ephemeral information, if any.*
   - $m \in \mathcal{M} \cup \{\perp, \dashv\}$ *is the next message to be sent in the conversation, where* $\dashv$ *signifies that no further message needs to be sent.*
   - $\delta \in \{\mathsf{accept}, \mathsf{reject}, *\}$ *indicates* U*'s decision in the current conversation. The symbol* $*$ *signifies a decision has not yet been made. If* $\delta = \mathsf{reject}$ *is returned then* ϵ' *and* m *are set to* $\perp$ *and* κ *must be equal to* $*$*.*
   - $\kappa \in \mathcal{K} \cup \{*\}$ *is the secret session key computed, where* $*$ *denotes that it has not been computed yet.*

We often abuse notation and use the symbol Π to denote both the protocol function and the entire protocol (Setup, Π) and we assume that params is made implicit in the use of Π. See Figure 1 for a two round example; which will be the focus of this paper.

An unauthenciated key agreement protocol is said to be *correct* if when the messages are relayed faithfully, i.e. unmodified and in the correct order, between two participants, then they both accept and compute identical session keys, except with negligible probability over the randomness used in the algorithms.

In practice one defines a specific key agreement protocol by defining how each new input message is responded to, given the current player state ϵ. We implicitly assume that if the input state is $\perp$, then the output state and message are also $\perp$ and δ will be reject.

For such unauthenticated key agreement protocol the best security guarantee we can obtain is that of passive security. Such a protocol is said to be passively secure if a single session of the protocol does not leak any information regarding the computed session key to an arbitrary $\mathsf{poly}(\lambda)$-time adversary $\mathcal{A}$ that only eavesdrops on the conversation. For an unauthenticated key agreement protocol Π and an adversary $\mathcal{A}$, this is formalised in the $\mathsf{EAV\text{-}KA}$ experiment described in Figure 3. We denote $\mathcal{A}$'s advantage in the $\mathsf{EAV\text{-}KA}$ game as $\mathbf{Adv}^{\mathsf{EAV\text{-}KA}}_{\mathcal{A},\Pi}(\lambda) = \left|\frac{1}{2} - \Pr\left[\mathbf{Exp}^{\mathsf{EAV\text{-}KA}}_{\mathcal{A},\Pi}(\lambda) = 1\right]\right|$.

Definition 2 (Passive KA Security). *A key agreement protocol* Π *is* passively secure in the presence of an eavesdropper *if for all probabilistic* $\mathsf{poly}(\lambda)$*-time adversaries* $\mathcal{A}$*, the following conditions hold.*

1. *If messages are relayed faithfully by a benign adversary between two participant oracles, then both oracles accept holding identical session keys, and each participant's key is distributed uniformly at random over* $\mathcal{K}$*.*
2. *There exists a negligible function* $\mathsf{negl}(\lambda)$ *such that* $\boldsymbol{Adv}^{\mathsf{EAV\text{-}KA}}_{\mathcal{A},\Pi}(\lambda) \leq \mathsf{negl}(\lambda)$*.*

1. Two parties holding 1^λ execute protocol Π with one another. This results in a transcript tran of the entire conversation, and a key κ output by each of the parties.
2. A uniform bit $b \in \{0,1\}$ is chosen. If $b = 0$, set $\hat{\kappa} := \kappa$, and if $b = 1$ then sample $\hat{\kappa} \leftarrow_\$ \mathcal{K}$ uniformly at random.
3. $\mathcal{A}$ is given tran and $\hat{\kappa}$, and outputs a guess bit b'.
4. The output of the experiment is defined to be 1 if $b' = b$, and 0 otherwise.

Fig. 3: The EAV-KA Security Experiment $\mathbf{Exp}^{\mathsf{EAV\text{-}KA}}_{\mathcal{A},\Pi}(\lambda)$.

It is an easy exercise to see that our syntax captures the syntax of Diffie–Hellman, Frodo and NewHope. In addition it is another easy exercise to show that the standard unauthenticated Diffie–Hellman protocol meets our Passive KA Security defnition, assuming the Decision Diffie–Hellman problem is hard. In addition it is relatively easy to check that the proofs of security of the Frodo and NewHope key agreement schemes, given in [7,1], also imply security for our Passive KA definition.

Minor Active Security Properties We also introduce two simple active security notions relevant to KA protocols. Most well designed passive KA schemes are implicitly understood to satisfy these two notions, but we choose to make them explicit (with the definition of two new security experiments) as we shall require them later on.

The first of these formalises the notion of the first protocol message being sufficiently "unpredictable"; i.e. the adversary is not able to guess what the first message m_1 of the transcript tran is going to be. We define the M1-GUESS experiment in Figure 4 and denote an arbitrary adversary $\mathcal{A}$'s advantage in that game as

$$\mathbf{Adv}^{\mathsf{M1\text{-}GUESS}}_{\mathcal{A},\Pi}(\lambda) = \Pr\left[\mathbf{Exp}^{\mathsf{M1\text{-}GUESS}}_{\mathcal{A},\Pi}(\lambda) = 1\right].$$

1. One party holding 1^λ computes $(\epsilon', m_1, *, *) \leftarrow \Pi(\mathsf{params}, \mathsf{init}, \emptyset, \perp; \$)$.
2. $\mathcal{A}$ is given 1^λ and params and outputs a guess message m_1'.
3. The output of the experiment is defined to be 1 if $m_1' = m_1$, and 0 otherwise.

Fig. 4: The M1-GUESS Security Experiment $\mathbf{Exp}^{\mathsf{M1\text{-}GUESS}}_{\mathcal{A},\Pi}(\lambda)$.

The second security notion models the property that an adversary should not be able to obtain information about the final key κ even if it may choose the first protocol message. This definition applies only to two-messages KA protocols. To this intent, we define the experiment KEY-FORCE in Figure 5 and denote an arbitrary adversary $\mathcal{A}$'s advantage as

$$\mathbf{Adv}^{\mathsf{KEY\text{-}FORCE}}_{\mathcal{A},\Pi}(\lambda) = \Pr\left[\mathbf{Exp}^{\mathsf{KEY\text{-}FORCE}}_{\mathcal{A},\Pi}(\lambda) = 1\right]$$

1. The challenger sets 1^λ and runs Setup to obain params.
2. $\mathcal{A}$ is given 1^λ and params and ouputs a first message m_1.
3. If $m_1 \notin \mathcal{M}$ the experiment outputs 0. Otherwise, the challenger computes $(\perp, m_2, \delta, \kappa_0) \leftarrow \Pi(\mathsf{params}, \mathsf{resp}, \{m_1\}, \perp; \$)$, together with sampling $\kappa_1 \leftarrow\!\!\$\ \mathcal{K}$, from the KE key space.
4. A bit $b \leftarrow\!\!\$\ \{0,1\}$ is chosen uniformly at random.
5. $\mathcal{A}$ is given κ_b and returns a guess $\tilde{b}$.
6. The experiment outputs 1 if and only if $\tilde{b} = b$, and 0 otherwise.

Fig. 5: The KEY-FORCE Security Experiment $\mathbf{Exp}^{\mathsf{KEY\text{-}FORCE}}_{\mathcal{A},\Pi}(\lambda)$.

3 Forward-secure Authenticated Key Agreement Protocols and Security Model

In this section, we focus on the formal definition of Authenticated Key Agreement (AKA) protocols and the security model which we will use. For our purposes, we reformulate slightly Kudla's BJM and mBJM models [23] which were themselves an elaboration of Bellare and Rogaway's original model [4] and of Blake-Wilson *et al.*'s formulation for the public-key setting [6]. In particular, we add the appropriate elements so that forward secrecy is captured by our model.

First we present the definition of a general authenticated key agreement protocol. Then we describe the execution environment of our security model which is the first step in capturing forward secrecy. Next we present the security experiment and definition for mutual authentication. Finally, we present the security experiment and definition for a secure authenticated key agreement protocol which combines both mutual authentication and secrecy of session keys. We also include a discussion regarding the security notions, including forward secrecy, that this definition of security guarantees.

3.1 AKA Protocol Definition

The key difference, between the AKA protocol we will discuss in this section and the definition of a simpler key agreement protocol from Section 2.2, lies in the fact that AKA protocols hope to achieve *entity authentication*. That is, the parties seek to confirm each other's identities as well as establish a secret session key. To do so, we require the introduction of long-term keying material that belongs to specific entities which we then use in the computation of the protocol messages. We therefore modify Definition 1 as follows.

Definition 3 (AKA Protocol). *An* authenticated key agreement *(AKA)* protocol *is a triple of probabilistic* $\mathsf{poly}(\lambda)$*-time algorithms* $(\mathsf{Setup}, \mathsf{KGen}, \Pi)$ *such that:*

1. *The* setup *algorithm* Setup *functions similarly to the eponymous algorithm of a key agreement protocol.*
2. *The* key-generation *algorithm* KGen *takes as input the public parameters* params *and an entity identifier* U *and outputs an entity-specific public/ private key pair* $(\mathsf{pk}_U, \mathsf{sk}_U)$.

3. *The* protocol function Π *functions similarly to the function of a key agree-ment protocol with the following differences. It is of the form* $(\epsilon', m, \delta, \kappa) \leftarrow \Pi(\mathsf{params}, (U, \mathsf{pk}_U, \mathsf{sk}_U), \rho, (V, \mathsf{pk}_V), \tau, \epsilon; \$)$ *where:*
   - U *is the identifier of a participating entity and is the sender of a message. We write* $(\mathsf{pk}_U, \mathsf{sk}_U)$ *for the public/private key pair of entity* U.
   - V *is the identifier of a participating entity and is the intended recipient of* U*'s message. We write* pk_V *for the public key of*V.
   - *All other elements are as in Definition 1.*

Again *correctness* requires that whenever messages are relayed faithfully between two participants, then they both accept and compute identical session keys (except with negligible probability over the randomness used in the algorithms).

Similarly to key agreement protocols, we will usually present protocols by giving the flows of a single run. A description of the function Π can be easily inferred. Also, we will use abuse notation and write Π (or sometimes Σ) both for the protocol function and the entire protocol which includes key generation, i.e. for $(\mathsf{Setup}, \mathsf{KGen}, \Pi)$ (or sometimes $(\mathsf{Setup}, \mathsf{KGen}, \Sigma)$).

3.2 Execution Environment

In the BJM model, the challenger simulates to the adversary an execution environment which constitutes of several participants. We wish to obtain "active" security, and so we allow the adversary to be active in the running of the protocol between the different entities In particular communication between protocol participants, modelled as oracles, which are controlled by the adversary, i.e. it can choose to invoke oracles to send legitimate messages or to insert is own, as well as modify, redirect, delay or erase messages. Each oracle, at the command of the adversary, may engage in several concurrent sessions of the protocol, with the same partner or not.

Oracle Participants: As mentioned above, we model protocol participants as oracles which we assume run as probabilistic $\mathsf{poly}(\lambda)$-time algorithms. More precisely, all participating entities are grouped in a set $\mathcal{U}$ of identifiers (IDs), and each session (or "run") of the protocol is modelled by an oracle $\Pi^s_{U,V}$. This represents a participant $U \in \mathcal{U}$ believing it is engaging in a protocol session with $V \in U$ for the s-th time; we say that V is U's *intended partner*. Each participant $U \in \mathcal{U}$ possesses a public and private key pair $(\mathsf{pk}_U, \mathsf{sk}_U)$, generated by KGen, and which we assume is authenticated by some public-key infrastructure (PKI). Each oracle instance of U has access to both keys, and every oracle in the model has access to every other user's public key.

Each individual oracle $\Pi^s_{U,V}$ maintains a public transcript $T^s_{U,V}$ which it updates as follows. When it recieves a message m, it records it on $T^s_{U,V}$ and then invokes the protocol function on the corresponding input. When the function produces an output, this is also recorded on $T^s_{U,V}$ before being returned to the adversary. Each oracle $\Pi^s_{U,V}$ also maintains an internal decision state $\delta^s_{U,V}$. This decision may take one of four values:

- $*$: the initial state of the oracle which indicates it has not yet reached a decision.

- accept: the oracle has successfully terminated this run of the protocol after having computed some session key $\mathsf{k}^s_{U,V}$.
- reject: the oracle has terminated without computing a session key.
- revealed: the oracle had previously accepted and has since been revealed by the adversary, as is described below.

As indicated above, each oracle $\Pi^s_{U,V}$ maintains a variable $\mathsf{k}^s_{U,V}$ which holds the value $*$ until the protocol returns a computed session key. Finally, each oracle is also associated a role $\rho^s_{U,V} \in \{\mathsf{init}, \mathsf{resp}\}$ depending on its function in the protocol session. Within this model, the adversary $\mathcal{A}$ is represented as a $\mathsf{poly}(\lambda)$-time algorithm that interacts with the oracles via specific queries; in addition, it also has access to the public key of each participant together with the transcript of each oracle.

Oracle Queries: During a security experiment for AKA security, run by a challenger $\mathcal{C}$ simulating protocol participants as oracles to an adversary $\mathcal{A}$, the adversary can make various queries of the oracles, to which the challenger simulates the responses.

At the beginning of the experiment, $\mathcal{C}$ generates protocol-specific parameters params by running $\mathsf{Setup}(1^\lambda)$, $\mathcal{C}$ is also responsible for generating a set of participant IDs $\mathcal{U}$, where $|\mathcal{U}| = n_P$ and $n_P = \mathsf{poly}(\lambda)$. For each participant $U \in \mathcal{U}$, $\mathcal{C}$ then runs $\mathsf{KGen}(\mathsf{params})$ in order to generate a key pair $(\mathsf{pk}_U, \mathsf{sk}_U)$. The challenger $\mathcal{C}$ also imposes the constraint that a given participant $U \in \mathcal{U}$ can engage in at most n_S sessions with another given participant $V \in \mathcal{U}$, where $n_S = \mathsf{poly}(\lambda)$. Therefore, the model composes of the following set of oracles $\left\{\Pi^s_{U,V} \mid U, V \in \mathcal{U}, s \in [n_S]\right\}$. Finally, $\mathcal{C}$ initialises an empty list $\Gamma \leftarrow \emptyset$ which he will use to keep track of which participant oracles have been corrupted by the adversary as is explained below. The adversary $\mathcal{A}$ is then given params, $\mathcal{U}$ and $\{\mathsf{pk}_U\}_{U \in \mathcal{U}}$, and proceeds by making the following queries:

- **Send**$(\Pi^s_{U,V}, m)$: The requests $\mathcal{C}$ to send the message m to $\Pi^s_{U,V}$. The message is recorded on $T^s_{U,V}$ and $\mathcal{C}$ responds to the message according to the protocol, simulating user U interacting with V for the s-th time. If $m = \vdash$, then $\Pi^s_{U,V}$ initiates a new protocol run, and its role is set as $\rho^s_{U,V} \leftarrow \mathsf{init}$. If an oracle's first recieved message is any message other than $\vdash$, then it sets $\rho^s_{U,V} \leftarrow \mathsf{resp}$.
  Once the response to m is computed according to the protocol, it is added to $T^s_{U,V}$ before being returned to the adversary. If this response is $\dashv$, this is also recorded on the transcript.
- **Reveal**$(\Pi^s_{U,V})$: This query is used by $\mathcal{A}$ to request the session key computed by $\Pi^s_{U,V}$. If $\delta^s_{U,V} = \mathsf{accept}$, and hence $\mathsf{k}^s_{U,V}$ exists, then this is output and returned to $\mathcal{A}$. Otherwise, this query returns $\perp$. If the query is successful, $\delta^s_{U,V} \leftarrow \mathsf{revealed}$ and we say that this session has been *revealed*.
- **Corrupt**$(U, \mathsf{pk}'_U, \mathsf{sk}'_U)$: This allows $\mathcal{A}$ to request the long-term secret key of participant U and is able to replace U's key pair with one of its choice. The challenger $\mathcal{C}$ returns sk_U to $\mathcal{A}$ and replaces $(\mathsf{pk}_U, \mathsf{sk}_U)$ with $(\mathsf{pk}'_U, \mathsf{sk}'_U)$. All oracles in the simulation are updated with the new public key, and secret key for the oracles representing U. Such a participant U is called *corrupted* and C updates the set $\Gamma \leftarrow \Gamma \cup \{U\}$.

3.3 Secure Mutual Authentication

Now that we have described the execution envrionment, we are able to define the first goal of an authenticated key exchange protocol, namely mutual authentication. This notion was first defined in the original BR model [4] using the concept of matching conversations. The concept of matching conversations is used to determine whether or not two oracles have engaged in a protocol session together (by means of the adversary relaying messages from one to the other).

Definition 4 (Matching conversation). *Suppose we are given the transcripts* $T^{s}_{U,U'} = \{\vdash, r_1, m_2, r_2, \ldots, m_j, r_j\}$ *and* $T^{s'}_{V,V'} = \{m'_1, r'_1, m'_2, r'_2, \ldots, m_k, r_k\}$ *such that*

- $m'_i = r_i$ *for* $i \geq 1$,
- $m_i = r'_{i-1}$ *for* $i \geq 2$,
- *for* j *even:* $r_j = \dashv$ *and* $k = j - 1$,
- *for* j *odd:* $r_k = \dashv$ *and* $k = j$,
- $U = V'$ *and* $U' = V$,

then we say that the oracles $\Pi^{s}_{U,U'}$ *and* $\Pi^{s'}_{V,V'}$ *have engaged in a* matching conversation. *We also sometimes say that* $\Pi^{s}_{U,U'}$ *and* $\Pi^{s'}_{V,V'}$ *are* matching (oracles).

Entity authentication is captured in the BR model using the **No-Matching** event which is triggered if an adversary manages to make an oracle accept without a matching oracle. Here we reformulate this as a security experiment to be consistent with the more modern way of defining security.

For an AKA protocol Π and an arbitrary $\mathsf{poly}(\lambda)$-time adversary $\mathcal{A}$, the AKA-AUTH experiment is defined in Figure 6. The intuition behind this experiment is the same as the one behind the No-Matching event; the aim of $\mathcal{A}$ is to make an oracle accept without having perfectly relayed the messages to and from its intended partner, and to do so without corrupting either parties.We denote $\mathcal{A}$'s advantage in the AKA-AUTH security game as

$$\mathbf{Adv}^{\mathsf{AKA\text{-}AUTH}}_{\mathcal{A},\Pi}(\lambda) = \Pr\left[\mathbf{Exp}^{\mathsf{AKA\text{-}AUTH}}_{\mathcal{A},\Pi}(\lambda) = 1\right].$$

1. $\mathsf{Setup}(1^\lambda)$ is run to obtain params.
2. The challenger $\mathcal{C}$ generates $\mathcal{U}$ and runs $\mathsf{KGen}(\mathsf{params}, U)$ for every $U \in \mathcal{U}$ to obtain key pairs $(\mathsf{pk}_U, \mathsf{sk}_U)$.
3. $\mathcal{A}$ is given params and $\{\mathsf{pk}_U\}_{U\in\mathcal{U}}$ and access to the participant oracles via the **Send**, **Reveal** and **Corrupt** queries. Eventually, $\mathcal{A}$ outputs a chosen session $\Pi^{s}_{U,V}$.
4. The output of the experiment is defined to be 1 if $\delta^{s}_{U,V} = \mathsf{accept}$, $U, V \notin \Gamma$ and there does not exists another oracle $\Pi^{s'}_{U',V'}$ which has had a matching converstation with $\Pi^{s}_{U,V}$.

Fig. 6: The AKA-AUTH Security Experiment $\mathbf{Exp}^{\mathsf{AKA\text{-}AUTH}}_{\mathcal{A},\Pi}(\lambda)$.

Definition 5 (Secure Mutual Authentication). *We say that an AKA protocol* $\Pi = (\mathit{Setup}, \mathsf{KGen}, \Pi)$ *is a* secure mutual authentication *protocol if, for any* $\mathsf{poly}(\lambda)$*-time adversary* $\mathcal{A}$*, the following hold.*

- ($\mathit{Matching\ Conversations} \Rightarrow \mathit{Acceptance}$.) *If two oracles* $\Pi^s_{U,V}$ *and* $\Pi^{s'}_{U',V'}$ *have matching conversation, then both oracles accept.*
- ($\mathit{Acceptance} \Rightarrow \mathit{Matching\ Conversations}$.) *For all probabilistic* $\mathsf{poly}(\lambda)$*-time adversaries* $\mathcal{A}$*, there exists a negligible function* $\mathsf{negl}(\lambda)$ *such that* $\mathbf{Adv}^{\mathsf{AKA\text{-}AUTH}}_{\mathcal{A},\Pi}(\lambda) \leq \mathsf{negl}(\lambda)$.

3.4 Session Key Secrecy and Forward Secrecy

Given a definition for mutual authentication, we now need to provide a security definition for the main purpose of key agreement; namely agreeing a private key. The secrecy game we present in Figure 7 can be seen as an extension of the AKA-AUTH experiment used to define mutual authentication. This is a natural progression as intuitively, it makes sense for an AKA protocol to first authenticate the entity it is conversing with before establishing a shared secret session key. The game is played between a challenger simulating the AKA protocol Π and an arbitrary $\mathsf{poly}(\lambda)$-time adversary $\mathcal{A}$.

Before we formally define the secrecy game, we will describe it briefly here so that the following definition may be placed into some context. As explained above, the simulator $\mathcal{C}$ first sets up the participants and then allows $\mathcal{A}$ to interact with them using some of the three queries. At some point in the simulation, $\mathcal{A}$ then has to select a session on which it wishes to be *tested*. At that point, the simulator flips a coin and either returns to the adversary that session's true key or a newly randomly sampled one. The adversary then continues the game, with further access to the oracles as before. Eventually the adversary has to guess which key the simulator returned; the real one or a random one. If it guesses correctly, it wins the security game.

We allow the adversary to reveal keys of completed sessions as well as corrupt participants and therefore we must make sure that it does not ask to be tested on a session for which it could have trivially obtained the key. Sessions on which we allow the adversary to request a test are those which are called *fresh* as defined below.

Definition 6 (Fresh Session). *A protocol session, represented by an oracle* $\Pi^s_{U,V}$ *is called* fresh *if the following conditions hold:*

- $\Pi^s_{U,V}$ *has accepted, and therefore holds a computed session key, but has not been revealed; i.e.* $\delta^s_{U,V} = \mathit{accept}$ *and* $\mathsf{k}^s_{U,V} \neq *$.
- *Neither* U *nor his intended partner* V *has been corrupted by* $\mathcal{A}$*; i.e.* $U, V \notin \Gamma$.
- *There does not exist an oracle* $\Pi^{s'}_{U',V'}$ *which matches* $\Pi^s_{U,V}$ *and has been revealed.*

Note that this definition does not require $\Pi^s_{U,V}$ to have a matching partner. The session is still considered to be fresh even if the adversary has managed to make $\Pi^s_{U,V}$ accept by generating and sending its own message. In addition an oracle only needs to be fresh at the point of it being tested; after testing the adversary can corrupt the parties in a test session; thus capturing forward secrecy. The only restriction on future operations is that it may not pass a reveal query to a test session (or an oracle with a matching conversation).

The Secrecy Experiment: Security for AKA protocols in the BJM model is defined in terms of the experiment shown in Figure 7, run with an AKA protocol Π and an arbitrary $\mathsf{poly}(\lambda)$-time adversary $\mathcal{A}$. We denote $\mathcal{A}$'s advantage in the AKA-SEC security game as

$$\mathbf{Adv}^{\mathsf{AKA\text{-}SEC}}_{\mathcal{A},\Pi}(\lambda) = \left| \frac{1}{2} - \Pr\left[\mathbf{Exp}^{\mathsf{AKA\text{-}SEC}}_{\mathcal{A},\Pi}(\lambda) = 1\right]\right|$$

1. $\mathsf{Setup}(1^\lambda)$ is run to obtain params.
2. The challenger $\mathcal{C}$ generates $\mathcal{U}$ and runs $\mathsf{KGen}(\mathsf{params}, U)$ for every $U \in \mathcal{U}$ to obtain key pairs $(\mathsf{pk}_U, \mathsf{sk}_U)$.
3. $\mathcal{A}$ is given params and $\{\mathsf{pk}_U\}_{U\in\mathcal{U}}$ and access to the participant oracles via the **Send**, **Reveal** and **Corrupt** queries. Eventually, $\mathcal{A}$ outputs a chosen session $\Pi^s_{U,V}$.
4. If $\Pi^s_{U,V}$ is not fresh, it is rejected and $\mathcal{A}$ must submit a new one. If it is, $\mathcal{C}$ selects a bit $b \in \{0,1\}$ at random. If $b = 0$, set $\widehat{\mathsf{k}} = \mathsf{k}^s_{U,V}$, and if $b = 1$, then sample $\widehat{\mathsf{k}} \leftarrow_\$ \mathcal{K}$ uniformly at random.
5. $\mathcal{A}$ is given $\widehat{\mathsf{k}}$, as well the same information as before, and it may continue to interact via the **Send**, **Reveal** and **Corrupt** queries with the exception that it may not reveal the session on which it chose to be tested, nor any session with a matching conversation. It may however corrupt either of the participants that took part in that session. Eventually, $\mathcal{A}$ outputs a guess bit b'.
6. The output of the experiment is defined to be 1 if $b' = b$, and 0 otherwise.

Fig. 7: The AKA-SEC Security Experiment $\mathbf{Exp}^{\mathsf{AKA\text{-}SEC}}_{\mathcal{A},\Pi}(\lambda)$.

3.5 Full Security Definition

We finally combine both the notions of mutual authentication and session key secrecy into a single security definition. As mentioned briefly above, the most notable characteristic of our security defintion for AKA protocols is that it captures the property known as *forward secrecy*. This property requires that the compromise of long-term secret keying information of entities does not allow an adversary to obtain any information regarding past session keys that these entities might have established.

This is captured in our model since the adversary is allowed, before it makes its final guess, to submit a **Corrupt** query on the entities that took part in the test session. With that possibility in mind, we still require that its advantage in the AKA-SEC experiment remains negligible. Thus, proving that an AKA protocol satisfies our deifnition of security also proves that it possesses forward secrecy, in which case we say it is a *forward secure AKA protocol*. Additionally, our definition also captures the usual security properties of AKA protocol such as session-key reveal secrecy and third-party compromise security.

Definition 7 (Active AKA Security). *An authenticated key agreement protocol Π is* actively secure *if for all probablisitic $\mathsf{poly}(\lambda)$-time adversaries $\mathcal{A}$, the following conditions hold.*

1. *If messages are relayed faithfully (by a benign or an active adversary) between two participant oracles, then both oracles accept holding identical session keys, and each participant's key is distributed uniformly at random over $\mathcal{K}$.*
2. *Π is a secure mutual authentication protocol.*
3. *There exists a negligible function $\mathsf{negl}(\lambda)$ such that $\mathbf{Adv}^{\mathsf{AKA\text{-}SEC}}_{\mathcal{A},\Pi}(\lambda) \leq \mathsf{negl}(\lambda)$.*

4 A New AKA Protocol Construction

We now present in more detail our new construction of a secure AKA protocol. We also state the theorems that establish secrecy for keys and the level of authentication that our protocol offers. The detailed proofs are in the full version of the paper.

The construction: Let $E = (\mathsf{Setup}_E, \mathsf{KGen}_E, \mathsf{Enc}, \mathsf{Dec})$ be a public-key encryption scheme. Let $M = (\mathsf{KGen}_M, \mathsf{Mac}, \mathsf{Vrfy})$ be a message authentication code such that its key space is $\mathcal{K}_M = \{0,1\}^{l(\lambda)}$ for some polynomial function l, and its KGen_M algorithm simply selects a key from $\mathcal{K}_M$ uniformly at random. Let $\Pi = (\mathsf{Setup}_\Pi, \Pi)$ be a two-round key agreement protocol and finally, let $H_1 : \{0,1\}^* \to \{0,1\}^{l(\lambda)}$ and $H_2 : \{0,1\}^* \to \{0,1\}^{h(\lambda)}$, where h is a polynomial function, be two key derivation functions. Using these elements, we construct the AKA protocol $\Sigma = (\mathsf{Setup}_\Sigma, \mathsf{KGen}_\Sigma, \Sigma)$ where:

1. Setup_Σ takes as input the security paramter 1^λ and outputs public parameters params_Σ which contain the parameters of the encryption scheme E output by $\mathsf{Setup}_E(1^\lambda)$ and the parameters of the KA protocol Π output by $\mathsf{Setup}_\Pi(1^\lambda)$.
2. KGen_Σ takes as input params_Σ and an indentifier U. It then outputs a public/private key pair for U by setting $(\mathsf{pk}_U, \mathsf{sk}_U) \leftarrow \mathsf{KGen}_E(\mathsf{params}_E)$, i.e. a normal public-key encryption scheme key pair.
3. Σ functions as specified by the protcol run described in Figure 2. The protocol works by first wrapping the message flows, m_1 and m_2, of the unauthenticated key agreement in encryptions to each party and then sending a MAC tag on the identities under a key derived from the key agreement session key using the KDF H_1. The final AKA session key is derived from the underlying agreed key and the party identities, using a different KDF H_2.

Security of our scheme: Authentication of Bob to Alice is obtained by Bob prefixing the plaintext m_1 to his response m_2 in the second message flow $\mathfrak{m}_2$. In this way Alice can verify that the message m_1' that she receives is identical to the one she sent out, i.e. m_1, and therefore Bob must have decrypted it; since only Bob has Bob's decryption key. Authentication of Alice to Bob is obtained by Alice sending a valid MAC on the identities under a key derived from the underlying unauthenticated key agreement scheme. Since only Alice can decrypt Bob's message m_2, only Alice could compute the underlying key agreement session key and therefore the associated MAC key. Notice that the these forms of authentication also imply liveness of the parties. The above intuition is formalized by the following theorem.

Theorem 1. *If Π is M1-GUESS-secure and KEY-FORCE-secure, E is 2-IND-CCA-secure and M is MAC-sFORGE-secure, then Σ is a secure mutual authentication protocol.*

Finally, we show that our construction yields a protocol that guarantee key secrecy.

Theorem 2. *If Π is EAV-KA-secure, M1-GUESS-secure and KEY-FORCE-secure, E is 2-IND-CCA-secure and M is MAC-sFORGE-secure, then Σ is AKA-SEC- secure.*

References

1. Erdem Alkim, Léo Ducas, Thomas Pöppelmann, and Peter Schwabe. Post-quantum key exchange - A new hope. In Thorsten Holz and Stefan Savage, editors, *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016.*, pages 327–343. USENIX Association, 2016.
2. Mihir Bellare, Alexandra Boldyreva, and Silvio Micali. Public-key encryption in a multi-user setting: Security proofs and improvements. In Bart Preneel, editor, *EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 259–274. Springer, Heidelberg, May 2000.
3. Mihir Bellare, Ran Canetti, and Hugo Krawczyk. A modular approach to the design and analysis of authentication and key exchange protocols (extended abstract). In *30th ACM STOC*, pages 419–428. ACM Press, May 1998.
4. Mihir Bellare and Phillip Rogaway. Entity authentication and key distribution. In Douglas R. Stinson, editor, *CRYPTO'93*, volume 773 of *LNCS*, pages 232–249. Springer, Heidelberg, August 1994.
5. Daniel J. Bernstein, Daira Hopwood, Andreas Hülsing, Tanja Lange, Ruben Niederhagen, Louiza Papachristodoulou, Michael Schneider, Peter Schwabe, and Zooko Wilcox-O'Hearn. SPHINCS: Practical stateless hash-based signatures. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part I*, volume 9056 of *LNCS*, pages 368–397. Springer, Heidelberg, April 2015.
6. Simon Blake-Wilson, Don Johnson, and Alfred Menezes. Key agreement protocols and their security analysis. In Michael Darnell, editor, *6th IMA International Conference on Cryptography and Coding*, volume 1355 of *LNCS*, pages 30–45. Springer, Heidelberg, December 1997.
7. Joppe W. Bos, Craig Costello, Léo Ducas, Ilya Mironov, Michael Naehrig, Valeria Nikolaenko, Ananth Raghunathan, and Douglas Stebila. Frodo: Take off the ring! Practical, quantum-secure key exchange from LWE. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 16*, pages 1006–1018. ACM Press, October 2016.
8. Colin Boyd, Yvonne Cliff, Juan Gonzalez Nieto, and Kenneth G. Paterson. Efficient one-round key exchange in the standard model. In Yi Mu, Willy Susilo, and Jennifer Seberry, editors, *ACISP 08*, volume 5107 of *LNCS*, pages 69–83. Springer, Heidelberg, July 2008.
9. Ran Canetti and Hugo Krawczyk. Analysis of key-exchange protocols and their use for building secure channels. In Birgit Pfitzmann, editor, *EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 453–474. Springer, Heidelberg, May 2001.
10. Léo Ducas, Alain Durmus, Tancrède Lepoint, and Vadim Lyubashevsky. Lattice signatures and bimodal Gaussians. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 40–56. Springer, Heidelberg, August 2013.
11. Marc Fischlin, Felix Günther, Benedikt Schmidt, and Bogdan Warinschi. Key confirmation in key exchange: A formal treatment and implications for TLS 1.3. In *IEEE Symposium on Security and Privacy, SP 2016, San Jose, CA, USA, May 22-26, 2016*, pages 452–469, 2016.
12. Craig Gentry and Michael Szydlo. Cryptanalysis of the revised NTRU signature scheme. In Lars R. Knudsen, editor, *EUROCRYPT 2002*, volume 2332 of *LNCS*, pages 299–320. Springer, Heidelberg, April / May 2002.

13. Oded Goldreich, Shafi Goldwasser, and Shai Halevi. Public-key cryptosystems from lattice reduction problems. In Burton S. Kaliski Jr., editor, *CRYPTO'97*, volume 1294 of *LNCS*, pages 112–131. Springer, Heidelberg, August 1997.
14. Felix Günther, Britta Hale, Tibor Jager, and Sebastian Lauer. 0-RTT key exchange with full forward secrecy. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part II*, volume 10211 of *LNCS*, pages 519–548. Springer, Heidelberg, May 2017.
15. Jeffrey Hoffstein, Nick Howgrave-Graham, Jill Pipher, Joseph H. Silverman, and William Whyte. NTRUSIGN: Digital signatures using the NTRU lattice. In Marc Joye, editor, *CT-RSA 2003*, volume 2612 of *LNCS*, pages 122–140. Springer, Heidelberg, April 2003.
16. Jeffrey Hoffstein, Jill Pipher, and Joseph H. Silverman. NTRU: A ring-based public key cryptosystem. In Joe Buhler, editor, *Algorithmic Number Theory, Third International Symposium, ANTS-III, Portland, Oregon, USA, June 21-25, 1998, Proceedings*, volume 1423 of *Lecture Notes in Computer Science*, pages 267–288. Springer, 1998.
17. Jeffrey Hoffstein, Jill Pipher, and Joseph H. Silverman. NSS: An NTRU lattice-based signature scheme. In Birgit Pfitzmann, editor, *EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 211–228. Springer, Heidelberg, May 2001.
18. Tibor Jager, Florian Kohlar, Sven Schäge, and Jörg Schwenk. Generic compilers for authenticated key exchange. In Masayuki Abe, editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 232–249. Springer, Heidelberg, December 2010.
19. Tibor Jager, Kenneth G. Paterson, and Juraj Somorovsky. One bad apple: Backwards compatibility attacks on state-of-the-art cryptography. In *NDSS 2013*. The Internet Society, February 2013.
20. Tibor Jager, Jörg Schwenk, and Juraj Somorovsky. On the security of TLS 1.3 and QUIC against weaknesses in PKCS#1 v1.5 encryption. In Indrajit Ray, Ninghui Li, and Christopher Kruegel:, editors, *ACM CCS 15*, pages 1185–1196. ACM Press, October 2015.
21. Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography*. Chapman and Hall/CRC Press, 2007.
22. Jonathan Katz and Moti Yung. Scalable protocols for authenticated group key exchange. *Journal of Cryptology*, 20(1):85–113, Jan 2007.
23. Caroline J. Kudla. *Special Signature Schemes and Key Agreement Protocols*. PhD thesis, Royal Holloway University of London, 2006.
24. Yong Li, Sven Schäge, Zheng Yang, Christoph Bader, and Jörg Schwenk. *New Modular Compilers for Authenticated Key Exchange*, pages 1–18. Springer International Publishing, Cham, 2014.
25. Richard Lindner and Chris Peikert. Better key sizes (and attacks) for LWE-based encryption. In Aggelos Kiayias, editor, *CT-RSA 2011*, volume 6558 of *LNCS*, pages 319–339. Springer, Heidelberg, February 2011.
26. Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In Henri Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 1–23. Springer, Heidelberg, May 2010.
27. Nikos Mavrogiannopoulos, Frederik Vercauteren, Vesselin Velichkov, and Bart Preneel. A cross-protocol attack on the TLS protocol. In Ting Yu, George Danezis, and Virgil D. Gligor, editors, *ACM CCS 12*, pages 62–72. ACM Press, October 2012.
28. Paul Morrissey, Nigel P. Smart, and Bogdan Warinschi. The TLS handshake protocol: A modular analysis. *Journal of Cryptology*, 23(2):187–223, April 2010.
29. Phong Q. Nguyen and Oded Regev. Learning a parallelepiped: Cryptanalysis of GGH and NTRU signatures. In Serge Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 271–288. Springer, Heidelberg, May / June 2006.