SAS-Based Authenticated Key Agreement

Sylvain Pasini and Serge Vaudenay

EPFL
CH-1015 Lausanne, Switzerland
http://lasecwww.ep¤.ch

Abstract. Key agreement protocols are frequently based on the Dif£e-Hellman protocol but require authenticating the protocol messages in two ways. This can be made by a cross-authentication protocol. Such protocols, based on the assumption that a channel which can authenticate short strings is available (SAS-based), have been proposed by Vaudenay. In this paper, we survey existing protocols and we propose a new one. Our proposed protocol requires three moves and a single SAS to be authenticated in two ways. It is provably secure in the random oracle model. We can further achieve security with a generic construction (e.g. in the standard model) at the price of an extra move. We discuss applications such as secure peer-to-peer VoIP.

1 The SAS-Based Authenticated Key Agreement Problem

Secure communication channels are usually set up by authenticated key agreement protocols. This can be performed by relying on a public-key infrastructure, e.g. based on RSA [**?**] or the Dif£e-Hellman protocol [**?**]. Clearly, this is not well suited to the advent of mobile ad-hoc communications where ephemeral or bootstrap connections are needed "at once": we certainly would not like to register a certi£cate to connect a PDA to a cell phone or to print to the neighbor available printer device. Secure communications can also be manually set up. For instance, peer-to-peer links using PGP can be set up by checking the digest of a public key over the telephone. Wireless devices can be securely connected by having the user to manually check a hashed value as well. To save the human user load, the string to be manually checked must be as short as possible. Recently, protocols based on Short Authenticated Strings (SAS) have been studied by Vaudenay [**?**]. It was shown how to design and analyze a protocol to authenticate an arbitrary string assuming that we can authenticate a short one over a dedicated secure channel. Those protocols are based on commitment schemes. It was also brie¤y proposed how to design message cross-authentication protocols, namely protocols to authenticate arbitrary strings in two ways.

A SAS-based Authenticated Key Agreement (AKA) protocol can be easily designed by running the Dif£e-Hellman protocol over an insecure channel, then by authenticating the digest of the protocol transcript using a SAS-based message cross-authentication protocol. This typically results in a 5-move protocol in addition to the bidirectional SAS transmission. In the present work, we show how to decrease the interaction cost. Namely, we design a generic construction which can use a 4-move protocol in addition to the bidirectional SAS exchange. This construction can rely on the standard model (without random oracles). We also design an optimal 3-move protocol which is provably secure (with tight reduction) in the random oracle model.[1]

2 Preliminaries

We adopt the security model from [**?**,**?**,**?**] based on the one from Bellare-Rogaway [**?**]. We consider a network of participants which are located at some nodes. A participant at node n is associated to a given identity ID_n. He locally maintains a database of (K_j, ID_j) pairs meaning that he can use the symmetric key K_j to securely communicate with ID_j in a private and authenticated way. Participants can run concurrent protocols. A protocol speci£es a sequence of steps which consist of receiving a message and sending a response. An internal short-term state keeps track on previously completed steps. Once the protocol is completed, the short-term state is removed. A protocol starts with some speci£ed inputs and an initial state (in terms of database content). It ends with some speci£ed outputs (or an error message) and a £nal state. The difference between an input (resp. output) and an initial (resp. £nal) state is that the adversary has control on the £rst one but not on the second one, except if the node was corrupted or some information leaked. Protocol instances on a node n are denoted by a unique tag π_n^i. (Note that the state of a protocol related to a given tag changes with time as new steps are made.)

Nodes can communicate through an insecure broadband channel. In addition, they have access to peer-to-peer narrowband channels which can be used to authenticate short messages. A node receiving a message from one of these channels is ensured that this message was sent at some time in the past by a node whose identity is speci£ed by the channel itself. In this paper, we concentrate on key agreement and cross-authentication protocols, so we assume that nodes share no prior exchanged keys.

[1] After the present paper was submitted, a preprint was posted by Laur, Asokan, and Nyberg [**?**]. This paper includes another 3-move protocol which is provably secure based on a generic commitment (e.g. in the standard model) but not optimal.

2.1 Adversarial Model

By default, the adversary is assumed to have a full control on which node makes a new step of a given protocol instance, on the insecure channel, can in¤uence the delivery of messages (without modifying them) over the authenticated channels, can choose the inputs of the protocols, and has access to the outputs. Occasionally, the adversary can violate the privacy of the internal state of a given node or even corrupt the node so that his behavior with respect to future runs of any protocol is no longer guaranteed. More formally, the adversary has access to the following oracles.

Launch. $\mathsf{launch}(n, \mathsf{role}, x)$ launches a new protocol instance on node n playing role (e.g. either Alice or Bob) with input x. It returns a new instance tag π_n^i. Note that the instance inherits of the current node state as its input state.

Send. $\mathsf{send}(\pi, y)$ sends an incoming message y to the instance π. It returns an outgoing message z, or the £nal output of the protocol if it completed.

Test. $\mathsf{test}(n, k, \mathsf{ID})$ tells whether (k, ID) is an entry of the database of node n. In practice, this oracle may be implemented by an active adversary trying to impersonate node n to communicate with ID. If the attempt succeeds, it means that k was the right key to use.

Remove. $\mathsf{remove}(n, \mathsf{ID})$ removes any (k, ID) entry in the database of node n. In practice, this oracle may be implemented by an adversary making denial-of-services attacks in the communication link between n and ID so that n decides not to trust this connection anymore and to remove it.

Reveal. $\mathsf{reveal}(n)$ reveals the full current state of node n. This models side channels or careless uses.

Corrupt. $\mathsf{corrupt}(n)$ injects a malicious code in node n so that its behavior is no longer guaranteed.

The *attack cost* is measured by

- the number Q of launched instances of Alice or Bob, i.e. the *online complexity*.
- the additional complexity C, i.e. the *of¤ine complexity*.
- the probability of success p.

We call *one-shot attacks* the attacks which launch only two instances in total, i.e. $Q = 2$.

By convention, we describe protocols by putting a *hat* on the notation for messages received by a node (i.e. inputs of the send oracle) which are not authenticated since they can differ from messages which were sent (i.e. outputs of the receive oracle) in the case of an active attack. A message m from a node of identity ID over an authenticated channel is denoted $\mathsf{authenticate}_{\mathsf{ID}}(m)$.

2.2 Key Agreement, Cross-Authentication, and Mutual Authentication

Authenticated key agreement. An Authenticated Key Agreement (AKA) protocol between Alice and Bob starts with no input, is independent from the current state, and ends with no output but a £nal state specifying an entry (k, ID) to be inserted in the database: Alice of identity ID_A ends with (k, ID_B) and Bob of identity ID_B ends with (k, ID_A). An attack is successful if a $\mathsf{test}(n, k, \mathsf{ID})$ query positively answered where n and ID correspond to nodes on which no reveal nor corrupt query was made. For simplicity, we do not consider attacks making Alice and Bob end on some inconsistent states. Namely, *mutual authentication* is assumed to be (implicitly or explicitely) made by further communications.

To construct AKA protocols, we use the following building blocks.

Message cross-authentication. A Message Cross-Authentication (MCA) protocol between Alice and Bob of identity ID_A and ID_B starts with inputs m_A and m_B and ends with outputs (m_B, ID_B) and (m_A, ID_A), respectively. An adversary is successful if some instance ended on an incorrupted node with a pair (m, ID) but no instance on the node of identity ID with input m was launched. Note that test, remove, and reveal oracles are not relevant in this case.

Message mutual-authentication. A Message Mutual-Authentication (MMA) protocol between Alice and Bob of identity ID_A and ID_B starts with inputs m_A and m_B and ends with outputs ID_B and ID_A, respectively. A honest run of an MMA protocol must have $m_A = m_B$. An adversary is successful if some instance on an incorrupted node started with any m and ended with any ID such that no instance on the node of identity ID with input m was launched. As for MCA protocols, test, remove, and reveal oracles are not relevant. Obviously, we can transform an MCA protocol into an MMA protocol by just checking that the output message is equal to the input one on both sides.

MCA from MMA. We can also transform an MMA protocol with at least one move over the insecure channel into an MCA protocol at the price of an extra move: Bob of identity ID_B £rst sends his input message m_B and Alice of identity ID_A initiates an MMA protocol with input $m_A||\hat{m}_B$ by sending m_A together with the £rst MMA protocol message. Bob then follows the MMA protocol with input $\hat{m}_A||m_B$. The £nal outputs of Alice and Bob are $(\hat{m}_B, \mathsf{ID}_B)$ and $(\hat{m}_A, \mathsf{ID}_A)$ respectively.

To compare protocols we focus on the number of message moves over the insecure channel and on the length of authenticated messages. Furthermore, a protocol with two equal SAS to be sent in both directions (called symmetric SAS) will be considered as better than a protocol with two SAS of similar length (but not necessarily equal) to be exchanged. Indeed, some authentication channels may provide symmetric authentication at no extra cost.

2.3 Equivocable Commitment and Random Oracle Commitment

In this paper, we consider (tag-based) equivocable commitment schemes as de£ned by two algorithms commit and open and three oracles setup, $\mathsf{simcommit}$, and $\mathsf{equivocate}$.

Setup. $K_P \leftarrow \mathsf{setup}$ generates a public key K_P to be used as a common reference string and a secret key K_S to set up the $\mathsf{simcommit}$ and $\mathsf{equivocate}$ oracles. The public key K_P is implicitly used by all other algorithms and oracles but omitted in the notations for simplicity.

Commit. $(c,d) \leftarrow \mathsf{commit}(m,r)$ generates a commit value c and a decommit value d for a key r with a tag m. We assume that the distribution of the generated c is independent from r: the commitment is *perfectly hiding*.

Open. $r \leftarrow \mathsf{open}(m,c,d)$ yields r if (c,d) is a possible output for $\mathsf{commit}(m,r)$.

Simcommit. $(c,i) \leftarrow \mathsf{simcommit}(m)$ simulates a commit value c for a tag m and produces extra information ξ to be used later. The distribution of c should be the same as for the distribution of c generated by any $\mathsf{commit}(m,r)$. It also creates a unique identi£er i (a nounce) and inserts (i,m,c,ξ) in a database. This oracle uses the secret key K_S and should be secured. Access to the database must be restricted to this oracle and $\mathsf{equivocate}$.

Equivocate. $d \leftarrow \mathsf{equivocate}(i,r)$ yields d such that $r = \mathsf{open}(m,c,d)$ where (i,m,c,ξ) is in the database of $\mathsf{simcommit}$. This entry is further removed. (Namely, a simulated c can be equivocated only once.)

Access to $\mathsf{simcommit}$ and $\mathsf{equivocate}$ oracles is restricted depending on the application. The normal usage of the commitment scheme should be limited to commit and open but we stress that our security model assumes that the adversary may cheat on *some* commitments by having access to $\mathsf{simcommit}$ and $\mathsf{equivocate}$ oracles. Indeed, our notion of equivocable commitment relates to the notion of *simulation-sound* commitment [**?**].

The *hiding game* between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$ runs as follows.

1. $\mathcal{C}$ runs setup and sends K_P to $\mathcal{A}$
2. $\mathcal{A}$ sends a tag m to $\mathcal{C}$
3. $\mathcal{C}$ commits to a random key with tag m and sends a commit value c to $\mathcal{A}$
4. $\mathcal{A}$ computes some r and sends it to $\mathcal{C}$
5. $\mathcal{C}$ releases a decommit value d and $\mathcal{A}$ wins if $r \leftarrow \mathsf{open}(m,c,d)$

In that case, the adversary has access to the $\mathsf{simcommit}$ and $\mathsf{equivocate}$ oracles but cannot query $\mathsf{simcommit}$ with the selected tag m. Since the commitment is perfectly hiding, no adversary can win this game with a probability larger than 2^{-k} where k is the length of the key r.

The *binding game* between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$ runs as follows.

1. $\mathcal{C}$ runs setup and sends K_P to $\mathcal{A}$
2. $\mathcal{A}$ sends a tag m and a commit value c to $\mathcal{C}$
3. $\mathcal{C}$ picks a random r and sends it to $\mathcal{A}$
4. $\mathcal{A}$ produces a decommit value d and wins if $r \leftarrow \mathsf{open}(m,c,d)$

We say that the commitment with k-bit keys r is (T,ε)-secure is any adversary with complexity limited to T has a wining probability of at most $2^{-k}+\varepsilon$. In that case, the adversary has access to the $\mathsf{simcommit}$ and $\mathsf{equivocate}$ oracles but cannot query $\mathsf{simcommit}$ with the selected tag m.

Secure equivocable commitment schemes can be easily constructed based on simulation-sound trapdoor commitments by MacKenzie-Yang [**?**] as detailed in [**?**]. Constructions can be in the standard model with a common reference string, e.g. based on the security of DSA signatures [**?**] or Cramer-Shoup signatures [**?**]. We can also build an ef£cient equivocable commitment scheme based on the random oracle model.

Random oracle commitment scheme. Let ℓ_c, ℓ_e, and k be three integers. The setup algorithm is unused, but we assume that we can use three oracles:

H. $c \leftarrow H(e,r,m)$ queried with an ℓ_e-bit string e and a k-bit string r, looks whether an entry (e,r,m,c) in a list exist. If not, the oracle creates one with a random ℓ_c-bit string c. In any case, the oracle answers c.

Simcommit. $(c,i) \leftarrow \mathsf{simcommit}(m)$ simply picks a random ℓ_c-bit string c and a nounce i and stores (i,c,m) in a list.

Equivocate. $d \leftarrow \mathsf{equivocate}(i,r)$ gets (i,c,m) and removes it form the list. The oracle then picks a random ℓ_e-bit string e. If $(e,r,m,\cdot)$ exists in the H list, the oracle fails. Otherwise, (e,r,m,c) is inserted. Clearly, if the number of oracle accesses to H and $\mathsf{simcommit}$ is limited by q, the probability that the oracle fails at least once is less than $q^2 \times 2^{-\ell_e-1}$.

The algorithm $\mathsf{commit}(m,r)$ simply picks e at random, queries $H(e,r,m)$ and outputs $d=(e,r)$. The algorithm $\mathsf{open}(m,c,d)$ simply checks that $H(d,m)=c$ and parses $d=(e,r)$ to yield r. Unless $\mathsf{equivocate}$ fails, this scheme is clearly an equivocable commitment scheme as previously de£ned. Since all commit values c are generated in an independent way, there are no collisions with probability at least $1 - q^2 \times 2^{-\ell_c - 1}$. Clearly, being able to decommit any c to two values would lead H to a collision. Hence, the scheme is $(q, 2^{-k} + q^2 \times 2^{-\ell_e - 1} + q^2 \times 2^{-\ell_c - 1})$-secure. In practice, $\mathsf{simcommit}$ and $\mathsf{equivocate}$ are unused. So, we can just instantiate H by a standard hash function, provided that instantiation of that kind of random oracle makes sense [**?**].

3 Previous SAS-Based Key Agreement Protocols

A classical authenticated Dif£e-Hellman [**?**] protocol over a multiplicative group spanned by a generator g consists, for Alice (resp. Bob) of picking a random integer x_A (resp. x_B), sending the Dif£e-Hellman public keys, $y_A = g^{x_A}$ (resp. $y_B = g^{x_B}$) over the authenticated channel, computing $z_A = y_B^{x_A}$ (resp. $z_B = y_A^{x_B}$) and ending with state (z_A, ID_B) (resp. (z_B, ID_A)). In this case, authenticated messages are pretty long, but authentication is necessary to thwart man-in-the-middle attacks.

We £rst informally present an AKA protocol from Hoepman [**?**]. It is based on the Dif£e-Hellman protocol and it uses an authenticated channel for the authentication of each Dif£e-Hellman value. This protocol runs in three steps: commitment, authentication, and opening. (The original protocol has a fourth step: the key validation.) Instead of revealing its Dif£e-Hellman public key, each party £rst commits on it, keeping it hidden. In the next step, each participant authenticates a piece of its Dif£e-Hellman public key. Finally, they open their commitments and check their respective commitment and authenticated string before completing the regular Dif£e-Hellman protocol.

Another AKA protocol, depicted on Fig. 1, was used by Zimmermann for the PGPfone in 1995[2]. Its advantage is to reduce the number of moves in the insecure channel and to make both authenticated strings equal. In this protocol, only the £rst participant Alice commits to its public key. The commitment is immediately opened when the other public key is received. Finally, the authenticated string is a piece of the digest (denoted truncH on Fig. 1) of the Dif£e-Hellman protocol transcript.

For both the Hoepman and the PGPfone protocols, the security is not formally proven ([**?**] only provides a sketch of argument for the security). Another approach consists of authenticating the transcript of a classical key agreement protocol by using an MMA protocol. The MANA protocols by Gehrmann-Mitchell-Nyberg [**?**,**?**,**?**] illustrates this. Finally, we study in what follows a generic construction reducing the amount of authenticated bits in AKA protocols. Using it with the Dif£e-Hellman key agreement protocol and the MCA protocol of [**?**], we obtain the DH-SC protocol of Čagalj-Čapkun-Hubaux [**?**].

[2] personal communication.

Alice		**Bob**
pick x_A, $y_A \leftarrow g^{x_A}$		pick x_B, $y_B \leftarrow g^{x_B}$
	$\xrightarrow{\text{commit to } y_A}$	
	$\xleftarrow{y_B}$	
$z_A \leftarrow \hat{y}_B^{x_A}$	$\xrightarrow{\text{open commitment}}$	$z_B \leftarrow \hat{y}_A^{x_B}$
$\mathsf{SAS} \leftarrow \mathsf{truncH}(y_A \| \hat{y}_B)$	$\xrightarrow{\mathsf{authenticate}_{\mathsf{Alice}}(\mathsf{SAS})}$	$\mathsf{SAS} \stackrel{?}{=} \mathsf{truncH}(\hat{y}_A \| y_B)$
check SAS is the same	$\xleftarrow{\mathsf{authenticate}_{\mathsf{Bob}}(\mathsf{SAS})}$	
£nal state: Bob, z_A		**£nal state**: Alice, z_B

Fig. 1. PGPfone 1995 Key Agreement Protocol.

Using an optimized MCA protocol we can save one protocol move. In what follows we describe the generic construction, analyze it, and study 3-move MMA and MCA protocols with symmetric SAS.

4 Reducing Key Agreement to Message Authentication

We can build an AKA protocol by exchanging Dif£e-Hellman keys through a message cross-authentication protocol.

We propose a generic SAS-based construction for an AKA protocol that we call the constructed AKA protocol or simply *the* AKA protocol. For this, we use an initial AKA protocol (with longer strings to be authenticated), that we call the AKA_0 protocol, and an MCA protocol with short SAS. Consider that the AKA_0 protocol requires $n_k \geq 2$ moves, the $n_k - 1$-th being from Alice to Bob, and the MCA protocol requires $n_a \geq 2$ moves over the insecure channel, the £rst one being from Alice to Bob. In the AKA protocol, the $n_k - 2$ £rst moves of the AKA_0 are performed over the insecure channel. Then, both participants assembles his view on the protocol transcript τ by concatenating all protocol messages (sent and received ones). Then, an MCA protocol starts. Alice wishes to authenticate τ concatenated with her $n_k - 1$-th message α in the AKA_0 protocol. Bob wishes to authenticate the same $\tau \| \alpha$ concatenated with his last message β in the AKA_0 protocol. (Note that Bob selects the message to be authenticated after receiving Alice's £rst message in the MCA protocol.) At the end, both participants use the authenticated messages to complete the AKA_0 protocol and end with £nal states as speci£ed in the AKA_0 protocol. We have $n_k + n_a - 2$ moves in total.

Note that MCA can have $n_a < 2$. (For instance the trivial MMA protocol exchanging authenticated digests has no move and thus we can build an MCA with only one move.) In that case, we augment the MCA protocol by virtual moves and we obtain n_k moves in total. However, MCA protocols with $n_a < 2$ must have pretty large SAS to exchange the messages.

We can make a similar construction based on an n'_a-move MMA protocol instead of an MCA protocol. In that case, we can only encapsulate the last move β of the AKA_0 protocol in the MMA protocol, leading us to $\max(n_k, n_k + n'_a - 1)$ moves in total.

Theorem 1. *Let us consider an n_k-move AKA protocol (the AKA_0 protocol) and an n_a-move MCA protocol. The generic construction is essentially an AKA protocol with* $\max(n_k, n_k + n_a - 2)$ *moves in which the structure of authenticated messages is similar as in the MCA protocol. There exists a constant μ such that for any T, if ε_1 resp. ε_2 denotes the best success probability of an adversary bounded by T against the AKA_0 protocol resp. the MCA protocol, then any adversary bounded by $T \times \mu$ against the AKA protocol has probability of success at most* $\varepsilon_1 + \varepsilon_2$.

Using the Diffie-Hellman protocol and an n_a-move MCA protocol leads us to a $\max(2, n_a)$-move AKA protocol in which the structure of authenticated messages is similar as in the MCA protocol. With the construction based on an MMA protocol, we obtain $\max(2, n'_a + 1)$ moves. In the case where we want to achieve small SAS, we must have $n_a \geq 2$, leading us to n_a moves using MCA protocols and $n'_a + 1$ moves using MMA protocols. Since $(n'_a + 1)$-move MCA protocols can be made from n'_a-move protocols, we may decrease the total number of moves in AKA protocols by starting from MCA protocols directly.

Proof. For each instance of Alice, we let τ_A be the constructed transcript of the $n_k - 2$ first messages in the AKA protocol and we let α_A be her last message, i.e. the $n_k - 1$-th message in the protocol. We further let $\hat{\tau}_B || \hat{\alpha}_B || \hat{\beta}$ be the accepted message from Bob at the end of the MCA protocol. Similarly, for each instance of Bob, we let τ_B be the constructed transcript of the $n_k - 2$ first messages in the AKA protocol, $\hat{\tau}_A || \hat{\alpha}_A$ be the accepted message at the end of the MCA protocol, and β be his last message in the AKA_0 protocol assuming that Alice's last one is $\hat{\alpha}_A$. We let $\alpha_B = \hat{\alpha}_A$. Bob's message to be authenticated is $\tau_B || \alpha_B || \beta$.

Given an adversary $\mathcal{A}$ against the AKA protocol, we construct a simulator $\mathcal{B}$ interacting with $\mathcal{A}$ and attacking the MCA protocol. We simply simulate instances running the AKA_0 protocol and launch the MCA protocol instances when appropriate. test, remove, reveal and corruct queries can easily be simulated. Clearly, the attack against the MCA protocol does not succeed with probability at least $1 - \varepsilon$. In those cases, we have $\tau_B = \hat{\tau}_B$, $\tau_A = \hat{\tau}_A$, $\alpha_A = \hat{\alpha}_A = \alpha_B = \hat{\alpha}_B$, and $\beta = \hat{\beta}$, just as if the instance of Alice and Bob had the AKA_0 protocol run over an authenticated channel.

We construct a simulator $\mathcal{C}$ interacting with $\mathcal{A}$ and attacking the AKA_0 protocol over an authenticated channel. The simulator simply replaces inputs to the send oracle by authenticated ones when possible, or fails, and simulates the MCA protocol. Clearly, running $\mathcal{A}$ in parallel with $\mathcal{B}$ and $\mathcal{C}$ with the same random source, we derive that whenever $\mathcal{A}$ succeeds, either $\mathcal{B}$ or $\mathcal{C}$ succeed. □

A trivial MMA protocol consists of authenticating the digest of the input message from a collision-resistant hash function. This protocol can be transformed into an MCA protocol by using 2 moves (to exchange m_A and m_B) plus the authentication of a SAS in two ways as for the construction in Section 2.2. We obtain a 2-move AKA protocol with symmetric SAS, but the length of the SAS is quite long (typically, 160 bits).

A SAS-based cross-authentication protocol was proposed in [**?**] by interleaving two SAS-based message authentication protocols. It is a 4-move MCA protocol with symmetric SAS and can thus be transformed into a 4-move AKA protocol with symmetric SAS based on Dif£e-Hellman.

5 A new SAS-Based Message Mutual-Authentication Protocol

We propose a new protocol improving the number of exchanged messages. As depicted on Fig. 2, and without any attack, Alice and Bob start with the same message, i.e. $m_A = m_B$. Each participant chooses a k-bit random value R_A and R_B, respectively. Alice starts by committing on her random value R_A by sending c, keeping it hidden. Bob sends the random value R_B. Then, Alice opens her value by sending the decommit value d. Finally, both authenticate the SAS which has been computed using a simple XOR function. Using the generic con-

Alice		Bob
input: m_A		**input**: m_B
pick $R_A \in_U \{0,1\}^k$		pick $R_B \in_U \{0,1\}^k$
$(c,d) \leftarrow \mathsf{commit}(m_A, R_A)$	$\xrightarrow{\quad c \quad}$	
	$\xleftarrow{\quad R_B \quad}$	
	$\xrightarrow{\quad d \quad}$	$\hat{R}_A \leftarrow \mathsf{open}(m_B, \hat{c}, \hat{d})$
$\mathsf{SAS} \leftarrow R_A \oplus \hat{R}_B$	$\xrightarrow{\mathsf{authenticate}_{\mathsf{Alice}}(\mathsf{SAS})}$	$\mathsf{SAS} \stackrel{?}{=} \hat{R}_A \oplus R_B$
check SAS is the same	$\xleftarrow{\mathsf{authenticate}_{\mathsf{Bob}}(\mathsf{SAS})}$	
output: Bob		**output**: Alice

Fig. 2. A New SAS-Based Message Mutual-Authentication Protocol.

struction with Di£e-Hellman we obtain a 4-move AKA protocol with symmetric SAS.

Theorem 2. *We consider adversaries against the MMA protocol of Fig. 2 who are bounded by complexity T, Q_A instances of Alice, and Q_B instances of Bob. We assume that we have an (T_C, ε)-secure equivocable commitment scheme. There exists a (small) constant μ such that any adversary wins either with probability limited to $Q_A \cdot (Q_A + Q_B)(2^{-k} + \varepsilon)$ or with complexity $T \geq T_C - \mu$.*

Proof. Any adversary which would attack an instance of either Alice or Bob needs one SAS to send her/him so that she/he can complete. This required SAS can easily be obtained from any instance of Alice since she does not need any prior authenticated message. It can also be obtained from any instance of Bob in which case he must be sent another SAS before. The output SAS by Bob is equal to the sent one. Indeed, a successful adversary interaction de£nes the £rst attacked instance and a prior sequence initiated by one instance of Alice followed by a chain (possibly empty) of instances of Bob and ended by the attacked instance. Every (unattacked) instance of Bob in this sequence is sending a SAS identical to the received one to the next instance. Every intermediate instance of Bob terminates with an output message which must be equal to the input message of the previous instance in the sequence (otherwise, they would be successfully attacked). However, the £nal instance in the sequence outputs a message which is different than the input of the previous instance. Hence, every instance in the sequence but the £nal one has the same input message and all instances yield the same SAS. Clearly, sending the output SAS from the leading Alice to the tailing instance produces a successful attack with no intermediate instance of Bob.

Let $\mathcal{A}_0$ be an adversary who launches at most Q_A instances of Alice and Q_B instances of Bob. We transform it into an adversary $\mathcal{A}$ who launches an instance of Alice and a single target instance (of either Alice or Bob) as follows:

1. $\mathcal{A}$ £rst picks two random numbers I, J such that $1 \leq I \leq Q_A$ and $1 \leq J < Q_A + Q_B$.
2. We initialize counters i and j to 0 and run $\mathcal{A}_0$ step by step.
   - Every time $\mathcal{A}_0$ would like to make a launch query to launch an instance of Alice, we increment i. If $i = I$, we really launch it and call the instance Alice π. Otherwise, we increment j and if $j = J$, we really launch it and call the target instance π'. Otherwise, we simulate the oracle call.
   - Every time $\mathcal{A}_0$ would like to make a launch query to launch an instance of Bob, we increment j. If $j = J$, we really launch it and call the target instance π'. Otherwise, we simulate the oracle call.
   - If we have to send a SAS to π, we just simulate the oracle call.

– If we have to send a SAS to π' and we already got a SAS from π which is equal to the expected one, we just send it. Otherwise, the attack fails.

Due to the previous discussion, if $\mathcal{A}_0$ succeeds, if π' is the £rst attacked instance for $\mathcal{A}_0$ and if π is the leading instance of Alice in the sequence, then $\mathcal{A}$ succeeds. Hence, the probability of success of $\mathcal{A}$ is at least $\frac{1}{Q_A(Q_A+Q_B-1)}$ times the probability of success p of $\mathcal{A}_0$.

We now have an adversary $\mathcal{A}$ with Alice and a target instance. We assume that the adversary complexity is bounded by $T_C - \mu$ for some constant overhead μ to be determined by the following reductions. We consider two cases: attacks targeting an instance of Bob and attacks targeting an instance of Alice. Let p_A resp. p_B be the probability of a target Alice resp. Bob and q_A resp. q_B be the success probability conditioned to both cases, respectively. The success probability of $\mathcal{A}$ is $p = q_A p_A + q_B p_B$ and we have $p_A + p_B = 1$.

In both cases, we de£ne a simulator $\mathcal{B}$ who simulates the two instances as follows. We £rst pick a random k-bit SAS. When an instance of Alice is launched for the £rst time by the adversary $\mathcal{A}$, we simulate a commitment c by using $\mathsf{simcommit}$. Then the corresponding $\hat{R}_B$ is sent to this instance of Alice, the commit value is equivocated so that it opens to the key $\mathsf{SAS} \oplus \hat{R}_B$. This simulation of Alice is perfect and has the property to determine the £nal SAS at the beginning. If the attack succeeds, the other instance will have to deal with a commit value with a different tag. Depending on whether the other instance is an Alice or a Bob, we simulate it so that we can win the hiding game or the binding game against a challenger $\mathcal{C}$ as depicted on Fig. 3. In the case of a target Alice,

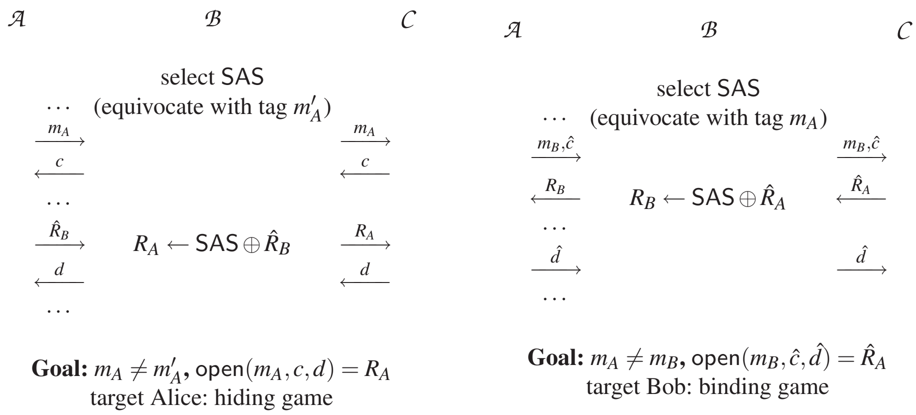

Fig. 3. Simulator Playing the Hiding/Binding Game.

the adversary succeeds if $\hat{R}_B$ leads the target instance to derive SAS. In that case we can correctly derive R_A and win the hiding game. Since the equivocable commitment is always perfectly hiding, we deduce $q_A = 2^{-k}$. We could have played the binding game in a trivial way and won with the same probability 2^{-k}. In the case of a target Bob, the adversary succeeds if $\hat{d}$ decommits to a key which leads Bob to the right SAS, thus to the key $\hat{R}_A$. In that case, we win the binding game with probability q_B. To summarize, we made an adversary playing the binding game with probability of success p. Therefore, $p \leq 2^{-k} + \varepsilon$. □

6 A new SAS-Based Message Cross-Authentication Protocol

We propose a new protocol based on the previous one, but improving the number of exchanged messages through the broadband insecure channel. Our protocol uses an almost strongly universal hash function family h [**?**,**?**]. In practice, one can use $h_K(x) = \mathsf{trunc}(\mathsf{hash}(K||x))$ where hash is a collision-resistant hash function and trunc truncates to the leading ρ bits. Our protocol also uses a commitment scheme to commit on a κ-bit key K. Contrarily to our previous protocol, the committed key K can now be pretty large. Using the generic construction with Dif£e-Hellman we obtain a 3-move AKA protocol with symmetric SAS. Note that we added an identity test on Alice's side to avoid trivial re¤ection attacks.

$$
\begin{array}{rcl}
\textbf{Alice} & & \textbf{Bob}\\
\textbf{input}: m_A & & \textbf{input}: m_B\\
\text{pick } K \in_U \{0,1\}^\kappa & & \text{pick } R \in_U \{0,1\}^\rho\\
(c,d) \leftarrow \mathsf{commit}(m_A,K) & \xrightarrow{\quad m_A,c \quad} & \\
 & \xleftarrow{\quad m_B,R \quad} & \\
 & \xrightarrow{\quad d \quad} & \hat{K} \leftarrow \mathsf{open}(\hat{m}_A,\hat{c},\hat{d})\\
\mathsf{SAS} \leftarrow \hat{R} \oplus h_K(\hat{m}_B) & \xrightarrow{\mathsf{authenticate}_{\mathsf{Alice}}(\mathsf{SAS})} & \mathsf{SAS} \stackrel{?}{=} R \oplus h_{\hat{K}}(m_B)\\
\text{check } \mathsf{SAS} \text{ is the same} & \xleftarrow{\mathsf{authenticate}_{\mathsf{Bob}}(\mathsf{SAS})} & \\
\text{check } \mathsf{Alice} \neq \mathsf{Bob} & & \\
\textbf{output}: \mathsf{Bob}, \hat{m}_B & & \textbf{output}: \mathsf{Alice}, \hat{m}_A
\end{array}
$$

Fig. 4. A New SAS-Based Message Cross-Authentication Protocol.

Theorem 3. *Let ℓ_e, ℓ_c be the parameters of the random oracle commitment scheme. Let q be the upper bound on the number of H queries. Let* $\varepsilon = q^2 2^{-\ell_e} +$

$q^2 2^{-\ell_c}$. Let h be an ε_h-almost strongly universal hash function family with ρ-bit digests, i.e. $\Pr[h_K(a) = \alpha, h_K(b) = \beta] \leq 2^{-2\rho} + 2^{-\rho}\varepsilon_h$ *for any* a, b, α, β *such that* $a \neq b$ *with a random K.*[3] *We consider adversaries against the message cross authentication protocol of Fig. 4 who are bounded by Q instances of Alice or Bob and by q queries to H. The success probability is limited by* $\frac{Q(Q-1)}{2}(2^{-\rho} + \varepsilon + \varepsilon_h)$.

By launching Q instances of either Alice or Bob with pairwise different input messages and by picking independent uniformly distributed $\hat{R}$, all SAS are independent and uniformly distributed so we have one matching with probability $1 - 2^{-Q\rho} \cdot 2^{\rho}!/(2^{\rho} - Q)!$ which is roughly $\frac{Q(Q-1)}{2} 2^{-\rho}$ when $Q \ll 2^{\frac{\rho}{2}}$. Hence, this bound is essentially tight. Note that the above attack can apply to any MCA protocol of similar structure (see [**?**]), so our protocol is optimal.

Proof. We let $\varepsilon_1 = \varepsilon_2 = \varepsilon_h$ (h is almost uniform). We have $\Pr[h_K(a) = \alpha] \leq 2^{-\rho} + \varepsilon_1$ and $\Pr[h_K(a) \oplus h_K(b) = \alpha] \leq 2^{-\rho} + \varepsilon_2$ for any a, b, α such that $a \neq b$ and with K uniformly distributed (h is ε-almost XOR universal [**?**]). In what follows, only those properties will be used. Namely, we could replace the condition on h by those two properties.

We de£ne a new character: the ¤ipped Bob who proceeds as Bob but £rst issues a SAS equal to $R \oplus h_{\hat{K}}(m_B)$ then receives a SAS for veri£cation. In a new protocol, Alice and the ¤ipped Bob can interact with two crossing SAS exchange.

We consider an adversary successfully running his attack with many instances for the original MCA protocol. We say that a given instance is attacked if it completed the protocol during which a SAS was received, with an output which is not consistent with the input of the instance who issued the received SAS. (Note that a successful adversary must have an attacked instance.) An attacked (target) instance (of either Alice or Bob) must receive one SAS from a (sending) instance. Note that those two instances must be different. (Indeed, no instance of Bob can send a SAS to himself otherwise it would have to be received before being sent. Similarly, no instance of Alice can accept a SAS coming from herself.) Clearly, both instances must agree on the SAS to complete. Hence, if the SAS sent by the target instance is forwarded to the sending instance then both instances fully interact. We can guess the pair of instances with probability $2/(Q(Q-1))$. Hence, we can simulate all instances except the two guessed ones. Since the SAS veri£cation phase is the last step on both instances, there is no trouble to make the two instances exchange their SAS. We thus transform the initial adversary against the MCA protocol with success probability p

[3] Note that this de£nition of almost strongly universal hashing is slightly different from [**?**,**?**] in the sense that perfect uniformity is not required.

into a one-shot adversary against our new protocol with success probability at least $2p(Q(Q-1))^{-1}$.

The interaction of the transformed adversary with an instance of Alice consists of two steps

A_1 sending her her message m_A (for the launch query) and getting her commit value c (for the £rst send query)

A_2 giving her Bob's alleged message $\hat{m}_B$ and random value $\hat{R}$ and getting her decommit value d.

Alice's SAS equals $\hat{R} \oplus h_K(\hat{m}_B)$ where K is the result of $\mathsf{open}(m_A, c, d)$. The second step must be performed after the £rst one.

The interaction of the adversary with an instance of Bob consists of two steps

B_1 sending him his message m_B (for the launch query) and Alice's alleged message $\hat{m}_A$ and commit value $\hat{c}$ and getting his random value R (for the £rst send query)

B_2 giving him Alice's alleged decommit value $\hat{d}$.

The adversary wins if the two instances complete and compute the same SAS and if the input message of one instance is different from the output message of the other instance.

In what follows we show that all cases can be simulated so that we can win a hard game, proving that the probability of success is at most $2^{-\rho} + \varepsilon + \max(\varepsilon_1, \varepsilon_2)$.

Cases Alice-Alice. We number 2 the instance of Alice whose A_2 step is the last. Since the commitment is perfectly hiding, this Alice leaks no information about K^2 (variable K for Alice number 2) until this very last step. Hence, K^2 is independent from the rest and $\hat{R}^1 \oplus \hat{R}^2 \oplus h_{K^1}(\hat{m}_B^1) = h_{K^2}(\hat{m}_B^2)$ with probability at most $2^{-\rho} + \varepsilon_1$.

On Bob's incoming $\hat{c}$ (Step B_1). In the random oracle commitment model, we only consider the event where no collision occurred. Hence, a commit value $\hat{c}$ issued by the adversary for an instance of Bob is either a real output by H and can only be opened in a single way, or no output from H. In the latter case, we can consider $(\perp, \perp, \perp, \hat{c})$ as a new entry in the H list and count it as an extra oracle call. This way, $\hat{c}$ can never be opened. Hence, with probability at least $1 - (q+1)(q+2)2^{-\ell_e - 1} - (q+1)(q+2)2^{-\ell_c - 1}$, which is larger than $1 - \varepsilon$, the commit value(s) $\hat{c}$ by the adversary are either openable in a single £xed way or not openable. If they are not openable, the adversary fails. If openable $\hat{c}$ are issued by an oracle call to H by the adversary, we can thus virtually replace the adversary release of $\hat{c}$ by an adversary release of $\hat{K}$ and step B_2 can be ignored. If openable $\hat{c}$ are issued by other oracle calls to H, it can only be by a simulation of Alice, leading us to $c = \hat{c}$, thus $\hat{K} = K$ and $m_A = \hat{m}_A$.

Cases Bob-Bob. We number 2 the instance of Bob whose B_1 step is the last one. Those cases produce no oracle calls to H by Alice, so $\hat{K}^1$ and $\hat{K}^2$ are selected by the adversary before the B_1^2 step. Note that R^1 is already released. The attack succeeds if $R^2 = R^1 \oplus h_{\hat{K}^1}(m_B^1) \oplus h_{\hat{K}^2}(m_B^2)$ where R^2 is independent of the righthand term and selected at random by the second Bob. Clearly, this succeeds with probability $2^{-\rho}$.

Cases Alice-Bob. Without loss of generality, we can assume that B_2 is the last step.

In cases $A_1A_2B_1B_2$, R is selected in step B_1 so the adversary succeeds with probability $2^{-\rho}$.

In cases $A_1B_1A_2B_2$ with $c \neq \hat{c}$ or in cases $B_1A_1A_2B_2$ (necessarily with $c \neq \hat{c}$), the adversary has no information about K until step A_2 and succeeds when $\hat{R} \oplus R \oplus h_{\hat{K}}(m_B) = h_K(\hat{m}_B)$. Hence succeeds with probability at most $2^{-\rho} + \varepsilon_1$.

In cases $A_1B_1A_2B_2$ with $c = \hat{c}$, we must have $m_A = \hat{m}_A$. This can only be an attack for $m_B \neq \hat{m}_B$. The adversary has no information about K until step A_2 and succeeds when $\hat{R} \oplus R = h_K(m_B) \oplus h_K(\hat{m}_B)$, hence with probability at most $2^{-\rho} + \varepsilon_2$. $\square$

With the same analysis as in [**?**], in a network of N participants, each limited to R runs of the protocol, and a maximal attack probability *at large p*, we should use $\rho \approx \log_2 \frac{N^2R^2}{2p}$. When p is the probability to attack a target node, we should use $\rho \approx \log_2 \frac{NR^2}{2p}$. With $N \approx 2^{20}$, $R \approx 2^{10}$, and $p \approx 2^{-10}$, we obtain $\rho \approx 49$. In an ATM-like environment, we can take $N = 2$, $R = 3$, and $p = 3 \cdot 10^{-4}$, leading us to $\rho \approx 15$. In between, we believe that $\rho = 20$ bits provides enough security in a small community of human users.

7 Conclusion

We have shown how to construct ef£cient SAS-based AKA protocols based on existing ones and SAS-based MMA or MCA protocols. We have proposed a new 3-move MMA protocol using a generic commitment scheme. It can make a secure and ef£cient SAS-based AKA protocol with 4 moves over the insecure channel. We have also proposed a new 3-move MCA protocol using random oracle commitments. It can make a secure and ef£cient SAS-based AKA protocol with 3 moves in the random oracle model. For both constructions, we can have e.g. a SAS of 20 bits. Note that our two constructions use the same authenticated strings in both directions.

Applications of such protocols can be traditional key agreement, but run in an ad-hoc way. For instance, it can be used to exchange PGP public keys to be authenticated by a human-to-human telephone conversation. It can also be used to secure peer-to-peer VoIP communications. Other straightforward applications can be the Bluetooth-like establishment of symmetric key between associated wireless devices, e.g. for wireless USB.