

An Extension to Bellare and Rogaway (1993) Model: Resetting Compromised Long-Term Keys^{*}

Colin Boyd¹ and Kim-Kwang Raymond Choo¹ and Anish Mathuria²

¹ Information Security Institute
Queensland University of Technology
GPO Box 2434, Brisbane, QLD 4001, Australia

² Dhirubhai Ambani Institute of Information and Communication Technology
Gandhinagar, Gujarat, India
boyd@isrc.qut.edu.au; raymond.choo.au@gmail.com;
anish_mathuria@da-iict.org

Abstract. A security proof in the Bellare–Rogaway model and the random oracle model is provided for a protocol closely based on one originally proposed by Boyd (1996), which enjoys some remarkable efficiency properties. The model is extended so that it can detect a known weakness of the protocol that cannot be captured in the original model. An alternative protocol, provably secure in the extended model and the random oracle model, offering the same efficiency features as the original protocol is proposed. Moreover, our alternative protocol provides key confirmation and forward secrecy. It also allows session keys to be renewed in subsequent sessions without the server’s further involvement even in the event that the long-term key or the earlier session key have been compromised.

1 Introduction

Protocols for key establishment are a foundational element in communications security. There has been an enormous amount of research effort expended in design and analysis of such protocols and yet there are still worthwhile contributions to be made even in the simple scenario of two users with an on-line server. For example, it is worthwhile to improve upon the performance cost associated with such protocols and ensure that the security goals can still be guaranteed.

Gong [14] has shown that protocols using timestamps require fewer messages and rounds than protocols using nonce-based challenge-response.

^{*} This is the full version of the conference proceedings that appears in 11th Australasian Conference on Information Security and Privacy - ACISP 2006, Lecture Notes in Computer Science, Springer-Verlag.

Boyd [8] proposed a novel method of achieving key freshness which does not require both participants' nonces to be passed to the server, thus reducing the number of messages and rounds to the same as that required for timestamp-based protocols. However, a known weakness of Boyd's protocol class is that if a user's long-term key is compromised, then an attacker can masquerade as that user even after the compromised key is replaced with a new one. Moreover, Boyd's protocol class does not have a proof of security; its purported security is based on heuristic arguments. The main problem with the heuristic approach is that it does not provide a clear framework for defining a "secure" protocol and what constitutes an "attack". Since this approach does not account for all possible attacks, the security guarantees are limited and often insufficient. In contrast, the provable security paradigm for protocols provides a formal foundation for defining a "secure" protocol and allows rigorous proofs of security to be developed.

In this paper we prove the original protocol of Boyd secure in the widely accepted model of Bellare and Rogaway (hereafter referred to as the BR93 model) [4]. In the BR93 model, there exists a powerful adversary who can interact with all the participants, with an aim to learn some information about one session key. Therefore, one tries to prove the indistinguishability of the session key (from a random key) for the adversary. The BR93 model has been further revised several times by several other researchers. However, like many other users of these models, we find that they are insufficiently rich to capture all reasonable actions of the adversary. In a practical system we may expect that once the compromise of a user has been detected, that user will be reset with a new long-term key and then allowed to continue working. In the type of protocols we are concerned with this scenario will allow the adversary to masquerade as that user. However, since there is no notion of resetting in the BR93 model there is no way to observe such a possibility. Therefore we extend the model to allow more capabilities for the adversary.

We then propose an equally efficient alternative protocol that provides protection against the compromise of long-term keys without taking recourse to revocation lists.

Contributions of Paper. The contributions of this paper are three-fold:

1. A revised protocol of Boyd [8] is proven secure in the BR93 model and the random oracle model (also known as the ideal hash model) [5]³.
2. The BR93 model is extended to allow more realistic adversary capabilities, under which the proven secure protocol of Boyd becomes insecure. Protocols proven secure in the extended model will also be secure in the original model.
3. An alternative protocol that is efficient in both messages and rounds is then shown to be secure in the extended BR93 model and the random oracle model. Furthermore, it provides key confirmation and forward secrecy⁴ and allows session keys to be renewed in subsequent sessions without the server's further involvement (i.e., re-authentication) even in the event that the long-term key or the earlier session key have been compromised. We remark that there are very few server-based protocols that achieve forward secrecy and allow re-authentication in the event that the long-term key or the earlier session key have been compromised.

Organization of Paper. Section 2 reviews the BR93 model and the mathematical preliminaries. Section 3 describes a protocol closely based on one originally proposed by Boyd [8] and provides a proof of its security in the BR93 model. Section 4 describes the limitation of the proof for the original protocol and extends the model so that there is capability to reset long-term keys. Section 5 describes an alternative protocol and provides a proof of its security in the extended model. A comparative summary is presented in Section 6. An extension to this alternative protocol allows session keys to be renewed in subsequent sessions without the server's further involvement even in the event that the long-term key or the ear-

³ Some might argue that a proof in the random oracle model is more of a heuristic proof than a real one. However, despite the criticism, this model is still widely accepted by the cryptographic community. We remark that recently, the first practical and provable-secure oblivious transfer password-based protocol whose proof of security relies on the random oracle model was recently published [11]. In many applications, a very efficient protocol with a heuristic security proof is preferred over a much less efficient one with a complete security proof [9]. Moreover, as Black [7] observed, no scheme has yet to be proven secure in the random-oracle model and broken once instantiated with some hash function, unless that was the goal from the very beginning.

⁴ When the long-term key of an entity is compromised the adversary will be able to masquerade as that entity in any future protocol runs. However, the situation will be even worse if the adversary can also use the compromised long-term key to obtain session keys that were accepted before the compromise. Protocols that prevent this are said to provide *forward secrecy*.

lier session key have been compromised is also described in this section. Section 7 presents the conclusions.

2 Provable Security Paradigm for Protocols

Bellare and Rogaway provide the first formal definition for a model of adversary capabilities with an associated definition of security (which we refer to as the BR93 model in this paper) in their 1993 paper [4] where they provide mathematical proofs for two-party entity authentication protocols. In the model, there exist a powerful adversary who can interact with all the participants, with an aim to learn some information about one session key. Therefore, one tries to prove the indistinguishability of the session key (from a random key) for the adversary.

2.1 The Adversarial Model

Informally the adversary, \mathcal{A} , is allowed to fully control the communication network by injecting, modifying, blocking, and deleting any messages at will. \mathcal{A} can also request for any session keys adaptively. The adversary interacts with a set of *oracles*, each of which represents an instance of a principal in a specific protocol run. Each principal has an identifier, U . An oracle, Π_U^s , represents the actions of principal U in the protocol run indexed by integer s . Formally, \mathcal{A} can adaptively query the following oracles, as follows:

- Send(U, s, m)** This query allows \mathcal{A} to make U runs the protocol normally. Π_U^s will return to \mathcal{A} the same next message that an honest principal, U , would if sent message m according to the conversation so far. If Π_U^s accepts the session key or halts this is included in the response. \mathcal{A} can also use this query to initiate a new protocol instance by sending an empty message m .
- Reveal(U, s)** This query models \mathcal{A} 's ability to find session keys. If a session key, K_s , has previously been accepted by Π_U^s , then it is returned to \mathcal{A} . An oracle can only accept a key once. An oracle is called *unfresh* if it has been the object of a **Reveal** query.
- Corrupt(U)** This query returns the oracle's long-term secret key. A principal is called *corrupted* if it has been the object of a **Corrupt** query. Note that this query does not return the session key since session keys can be learnt by the **Reveal** query or the entire internal state.
- Test(U, s)** Once Π_U^s has accepted a session key, K_s , \mathcal{A} can attempt to distinguish it from a random key as the basis of determining security

of the protocol. A random bit b is chosen; if $b = 0$, then K_s is returned while if $b = 1$ a random string is returned from the same distribution as session keys. This query is only asked once by \mathcal{A} .

2.2 Definition of Security

Definition of security in the BR93 model depends on the notion of the *partner* oracles to any oracle being tested. The way of defining partner oracles has varied in different papers using the model. Following recent trends, we define SID_U^s as the concatenation of all messages that oracle Π_U^s has sent and received.

Definition 1. *Two oracles are partnered if (1) they have accepted a session key with the same session identifier (SID), (2) each believes that the other is its partner, and (3) they agree on the initiator of the protocol.*

Definition 2 describes the freshness definition.

Definition 2. *An oracle Π_U^s is fresh at the end of its execution if (1) Π_U^s has accepted with partner Π_V^t (if such a partner exists), (2) Π_U^s and Π_V^t are unopened, and (3) principals U and V are uncorrupted.*

The security of the protocol is defined by the following game, \mathcal{G} , played between the adversary and an infinite collection of user oracles Π_U^s for $U \in \{U_1, \dots, U_Q\}$ and $s \in \mathbb{N}$ and server oracles Π_S^s . Firstly, long-lived keys are assigned to each user by running the key distribution algorithm \mathcal{K}_k on input of the security parameter k . Then, the adversary, $\mathcal{A}(1^k)$, is run. \mathcal{A} will interact with the oracles through the queries defined above. At some stage during the execution a **Test** query is performed by the adversary to a fresh user oracle. Eventually the adversary outputs a bit b' and terminates. Success of the adversary, \mathcal{A} , in this game is measured in terms of its *advantage* in distinguishing the session key of the **Test** query from a random key, i.e., its advantage in outputting $b' = b$. This advantage must be measured in terms of the security parameter k . If we define success to be the event that \mathcal{A} guesses correctly whether $b = 0$ or $b = 1$, then

$$\text{Adv}^{\mathcal{A}}(k) = |2 \cdot \Pr[\text{success}] - 1|.$$

To define validity of a protocol, we use the concept of a *benign adversary* as an adversary that faithfully relays flows between participants [4].

Definition 3. *A protocol P is a secure key establishment protocol if the following two properties are satisfied:*

Validity. *In the presence of a benign adversary partner oracles conclude with the same key except for a negligible probability.*

Indistinguishability. *For every probabilistic polynomial-time adversary, \mathcal{A} , the function $\text{Adv}^{\mathcal{A}}(k)$ is negligible.*

Security of a protocol is proved by finding a reduction to some well known computational problem whose intractability is assumed (i.e., in this paper, the Computational Diffie-Hellman (CDH) problem). In addition, we require the notion of an authenticated encryption scheme, which forms the basis of our proof for Protocol 2 described in Section 5.

2.3 The Computational Diffie-Hellman Assumption

Let $\mathbb{G} \in \mathbb{Z}_p^*$ be a cyclic group of prime order q and g is assumed to be a generator of \mathbb{G} , where \mathbb{G} is of prime order. The security parameters, p and q , are defined as the fixed form $q|p-1$ and $\text{ord}(g) = q$.

Computational Diffie-Hellman (CDH) Problem. Given an instance, (g, g^x, g^y) , output g^{xy} .

A Computational Diffie-Hellman (CDH) attacker, \mathcal{F}_{CDH} , in a finite cyclic group \mathbb{G} of prime order q with g as a generator, is a probabilistic machine, Δ , running in time t such that the success probability of \mathcal{F}_{CDH} when given random elements, $g^{N_1} \in \mathbb{G}$ and $g^{N_2} \in \mathbb{G}$ to output $g^{N_1 N_2} \in \mathbb{G}$, is less than ϵ , where the probability is over the random choice of N_1 and N_2 in \mathbb{Z}_q^* . In other words, the CDH assumption states that the success probability of \mathcal{F}_{CDH} for any $\frac{t}{\epsilon}$ is not too large.

2.4 Secure Authenticated Encryption Schemes

We now define the authenticated encryption scheme that will be employed in the protocol that we shall prove secure in Section 3.

Let k denote the security parameter. A *symmetric encryption scheme* $\mathcal{SE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ consists of three algorithms, namely: the *key generation* algorithm \mathcal{K} , the *encryption* algorithm \mathcal{E} , and the *decryption* algorithm \mathcal{D} as described below.

- \mathcal{K} is a probabilistic algorithm which, on input 1^k , outputs a key K .
- \mathcal{E} is a probabilistic algorithm which takes a key K and a message M drawn from a message space \mathcal{M} associated to K and returns a ciphertext C . This is denoted by $C \stackrel{R}{\leftarrow} \mathcal{E}_K(M)$.

- \mathcal{D} is a deterministic algorithm which takes a key K and a ciphertext C and returns the corresponding plaintext M or the symbol \perp which indicates an illegal ciphertext. This is denoted as $x \leftarrow \mathcal{D}_K(C)$. We require that $\mathcal{D}_K(\mathcal{E}_K(M)) = M$ for every $K \leftarrow \mathcal{K}(1^k)$.

For security we use the definitions of Bellare & Namprempre [2]. We require that the symmetric encryption scheme provides confidentiality in the sense of indistinguishability under chosen plaintext attacks (*IND-CPA security*) and provides integrity in the sense of preserving integrity of plaintexts (*INT-PTXT security*). We note that each of these is the weakest of the properties defined by Bellare and Namprempre and are provided by either encrypt-then-MAC or by MAC-then-encrypt constructions. Therefore there are many practical ways of implementing our protocol which can reasonably be expected to satisfy these assumptions. We now define these concepts more precisely.

For any efficient (probabilistic polynomial time) adversary, \mathcal{X} , the confidentiality security is defined in terms of the following game, which we call \mathcal{G}_1 .

1. The challenger chooses a key $K \leftarrow \mathcal{K}(1^k)$.
2. Given access to the encryption oracle, the adversary outputs two messages of equal length $M_0, M_1 \in \mathcal{M}$ of her choice.
3. The challenger computes $C_b \xleftarrow{R} \mathcal{E}_K(M_b)$ where $b \xleftarrow{R} \{0, 1\}$. The bit b is kept secret from the adversary.
4. The adversary is then given C_b and has to output a guess b' for b .

We define the advantage of the adversary, \mathcal{X} , playing the above game as

$$\text{Adv}_{\mathcal{X}}^{\text{ind-cpa}}(k) = |2 \cdot \Pr[b' = b] - 1|.$$

Definition 4. *The encryption scheme \mathcal{SE} is IND-CPA secure if the advantage of all efficient adversaries playing game \mathcal{G}_1 is negligible.*

For any efficient adversary, \mathcal{F} , the integrity security is defined in terms of the following game, which we call \mathcal{G}_2 .

1. Choose a key $K \leftarrow \mathcal{K}(1^k)$.
2. The adversary, \mathcal{F} is given access to the encryption oracle and also a *verification oracle* which on input a ciphertext C outputs 0 if $\mathcal{D}_K(C) = \perp$ and outputs 1 if C is a legitimate ciphertext.
3. The adversary wins if it can find a legitimate ciphertext C^* such that the plaintext $M = \mathcal{D}_K(C^*)$ was never used as a query to the encryption oracle. In this case we say the event *forgery* has occurred.

We define the advantage of the adversary playing the above game as

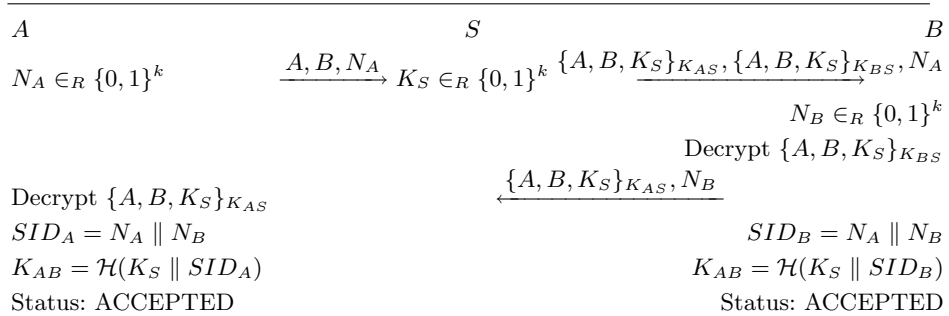
$$\text{Adv}_{\mathcal{F}}^{\text{int-ptxt}}(k) = |2 \cdot \Pr[\text{forgery}] - 1|.$$

Definition 5. *The encryption scheme \mathcal{SE} is INT-PTXT secure if the advantage of all efficient adversaries playing game \mathcal{G}_2 is negligible.*

3 A Provably-Secure Revised Protocol of Boyd

3.1 The Revised Protocol of Boyd

Protocol 1 is a server-based protocol in which users A and B as well as the server S contribute to the key value. All parameter choices depend on a security parameter k . In Protocol 1, the following notations are used: $\{m\}_K$ denotes an authenticated encryption of some message m under symmetric key K ; S denotes a server who shares long-term symmetric keys K_{AS} and K_{BS} with A and B , respectively; N_A , N_B , and K_S denote nonces generated by A , B and S , respectively; and \mathcal{H} is modelled as a random oracle. The session key obtained by A and B at the end of the protocol execution is denoted as K_{AB} .



Protocol 1: A revised key agreement protocol of Boyd

Protocol 1 is very similar to that proposed by Boyd [8]. Differences are as follows.

1. In the earlier protocol of Boyd, the session key is determined by a MAC function so that the session key is $K_{AB} = \text{MAC}_{K_S}(N_A, N_B)$.
2. There is no partnering mechanism (e.g., session identifiers) specified in the earlier protocol of Boyd. Message exchanges in the real world

are seldom conducted over secure channels. Therefore, it is realistic to assume that any adversary is able to modify messages at will, which is the case in the Bellare–Rogaway style models. As Goldreich and Lindell [12, Section 1.3] have pointed out, such an adversary capability means that the adversary is able to conduct concurrent executions of the protocol (one with each party). Therefore, without such partnering mechanism, communicating parties will be unable to uniquely distinguish messages from different sessions. Hence, in Protocol 1, we define partnership using the notion of session identifiers, SID ⁵.

3. The key confirmation messages have been removed, which consist of a handshake using the shared secret. These can easily be added in a standard way [3]. The session key itself must not be used to authenticate the key confirmation messages, otherwise the adversary can use them to easily distinguish the session key.

3.2 Security Proof

The proof follows that of Bellare and Rogaway [6] quite closely; differences include the use of a combined authenticated encryption scheme (as opposed to separate encryption and MAC functions) described in Section 2.4 and the different partnering function used. The general idea of the security proof is to assume that the protocol adversary can gain an advantage and use this to break the assumptions about the security of the encryption algorithm. Since the adversary relies on its oracles to run we simulate the oracles so that we can supply the answers to all the queries the adversary might ask. We cannot supply answers which rely on knowledge of the encryption keys that we are trying to break, so we use the integrity of plaintexts to show that these queries would, almost certainly, not be answered by any oracle running the protocol. As long as the simulation works with non-negligible probability the assumption about the encryption scheme fails.

Following Bellare and Rogaway [6] we need to extend the definition of a secure encryption scheme to allow the adversary to obtain multiple encryptions of the same plaintext under many different independent encryption keys. Such an adversary is termed a *multiple eavesdropper*. A multiple eavesdropper, \mathcal{ME} , is allowed to obtain encryptions of the same plaintext under two different independent encryption keys. We can

⁵ The security proof of Protocol 1 does not hinge on the difficulty of predicting a valid session identifier. In fact, we may assume that session identifiers are made publicly available when the status of the principal becomes “ACCEPTED”.

bound the advantage of a multiple eavesdropper by considering it as a special case of the multi-user setting analysed by Bellare, Boldyreva and Micali [1]. In their notation we have the case of $q_e = 1$, meaning that the adversary can only obtain one encryption for each encryption key. Specialising their main theorem gives the following.

Lemma 1. *Suppose that an adversary has advantage at most $\epsilon(k)$ for encryption scheme $(\mathcal{E}, \mathcal{D})$. Then a multiple eavesdropper has advantage not more than $n \cdot \epsilon(k)$.*

Notice that since an authenticated encryption scheme is also a secure encryption scheme in the sense defined by this result, it also holds for an authenticated encryption scheme. This allows us to define a variant of game \mathcal{G}_1 described in Definition 4 which we call \mathcal{G}'_1 . The only difference between these is that in \mathcal{G}'_1 the adversary is given access to two encryption oracles for two independently generated keys, and its challenge consists of two encryptions of either m_0 or m_1 under the two keys.

The idea of the proof is to consider the situation when the adversary at some stage *forges* a message successfully. When this occurs we can use the adversary to break the integrity of the authenticated encryption scheme. When this does not occur we use the adversary to break the confidentiality. More formally, define *forge* to be the event that the protocol adversary, \mathcal{A} produces a ciphertext C associated with an uncorrupted entity U such that $\mathcal{D}_K(C) \neq \perp$ where K is the long term key of entity U . Noting that $\Pr(\text{success}_{\mathcal{A}}) \leq \Pr(\text{forge}) + \Pr(\text{success}_{\mathcal{A}} | \overline{\text{forge}})$ we can split the proof up by showing that each of the two terms on the right is negligible.

3.2.1 Integrity Breaker Assume that \mathcal{A} is an adversary against the protocol. We use \mathcal{A} to construct a forger \mathcal{F} for the authenticated encryption scheme \mathcal{SE} described in Definition 4. We will say that the event $\text{success}_{\mathcal{F}}$ occurs if \mathcal{F} wins game \mathcal{G}_2 against \mathcal{SE} .

Lemma 2. *There is an efficient algorithm \mathcal{F} defined using \mathcal{A} such that if *forge* occurs with non-negligible probability then $\text{success}_{\mathcal{F}}$ occurs with non-negligible probability .*

In order to prove Lemma 2 we describe how \mathcal{F} is constructed. When \mathcal{F} runs it receives access to the encryption and verification oracles of the authenticated encryption scheme \mathcal{SE} . Its output must be a forged ciphertext for a message m which was not previously input to the encryption oracle.

In order to obtain the forgery \mathcal{F} runs \mathcal{A} by first choosing a user U_i for $i \in_R [1, Q]$. This user will be simulated as though its long-term key is the one used in \mathcal{SE} . For all other $j \in [1, Q]$ with $j \neq i$, \mathcal{F} generates the long-term shared key using the key generation algorithm \mathcal{K}_k . This allows \mathcal{F} to answer all the oracle queries from \mathcal{A} as follows.

- Send**(U, s, M) For any well-formed queries to S , \mathcal{F} can reply with valid ciphertexts, by choosing the session key and forming the ciphertexts, either directly using the known key or using the encryption oracle in the case of U_i . For queries to initiate a protocol run, \mathcal{F} can generate a random nonce and answer appropriately. Finally, consider a query to either an initiator or responder oracle including a claimed server message (corresponding to protocol messages 2 or 3). The relevant ciphertext can be verified either directly using the known key or using the verification oracle. If the ciphertext is verified correctly then the oracle accepts and this information is returned to \mathcal{A} .
- Reveal**(U, s) Since all the session keys are known from running the **Send** queries the query can be trivially answered with the correct session key (if accepted).
- Corrupt**(U) As long as $U \neq U_i$ all the private information is available and the query can be answered. In the case $U = U_i$ then the query cannot be answered and \mathcal{F} will abort and fail.
- Test**(U, s) Since all the accepted session keys are known from running the **Send**(U, s, M) queries the query can be trivially answered by identifying the correct session key.

\mathcal{F} continues the simulation until a forgery event against \mathcal{SE} occurs, or until \mathcal{A} halts. Note that as long as \mathcal{F} does not abort then the simulation is perfect. If **forge** occurs then the probability that the user involved is U_i equals $1/Q$. In this case the event **success $_{\mathcal{F}}$** occurs. Furthermore, in this case \mathcal{F} does not abort since U_i cannot be corrupted before the **forge** event. Therefore we arrive at the following upper bound.

$$\Pr(\text{forge}) \leq Q \cdot \Pr(\text{success}_{\mathcal{F}}) \quad (1)$$

3.2.2 Confidentiality Breaker Now assume that \mathcal{A} gains an advantage without producing a forgery. This time we use \mathcal{A} to form an algorithm \mathcal{X} which has a non-negligible advantage in the encryption scheme.

Lemma 3. *There is an efficient algorithm \mathcal{X} defined using \mathcal{A} such that if **success** occurs but **forge** does not occur, then \mathcal{X} wins game \mathcal{G}'_1 .*

Two random keys K and K' are chosen by the challenger for \mathcal{SE} and \mathcal{X} is given access to the encryption oracles for these keys. First \mathcal{X} chooses two users U_i and U_j for $i, j \in_R [1, Q]$. For all other $k \in [1, Q]$, \mathcal{X} generates the long-term key using the key generation algorithm \mathcal{K}_k . Next \mathcal{A} chooses two random session keys K_0 and K_1 . The two messages that \mathcal{X} asks of the challenger for \mathcal{SE} are $M_0 = (U_i, U_j, K_0)$ and $M_1 = (U_i, U_j, K_1)$. The challenger responds with a ciphertext pair C_b, C'_b which are the authenticated encryptions of either M_0 or M_1 under the two keys K and K' . Suppose that Q_S is the maximum number of `Send` queries that \mathcal{A} will ask of the server and Q_H is the maximum number of hash queries that \mathcal{A} will ask of the server. \mathcal{X} chooses a value s_0 randomly in $[1, Q_S]$. The idea is that \mathcal{X} will inject the ciphertexts C_b, C'_b into a random server `Send`(U, s, M) query. \mathcal{X} proceeds to simulate responses for \mathcal{A} as follows.

$\mathcal{H}(K||SID_i^k)$ For queries $\mathcal{H}(K||SID_i^k)$, if this query was asked before, then return the previous answer. Otherwise, return a random value, $v \in_R \{0, 1\}^k$. In addition, store this answer together with the query in a list of DH-tuples.

`Send`(U, s, M) First consider `Send` queries to the server. \mathcal{X} must simulate responses from the server with ciphertexts for two users A and B . There are three cases:

- Neither A nor B is equal to U_i . The session key K_S is chosen randomly by \mathcal{X} and the required tickets can be generated by \mathcal{X} using the long-term keys chosen.
- One of A or B is equal to U_i and the other is not equal to U_j . \mathcal{X} chooses the session key randomly and obtains the ticket for U_i using the encryption oracle and generates the ticket for the other user with the known long-term key.
- One of A or B is equal to U_i and the other is equal to U_j . If $s = s_0$ then \mathcal{A} uses C_b and C'_b as the two tickets for U_i and U_j . Otherwise \mathcal{X} chooses the session key randomly and obtains the tickets for U_i and U_j using the encryption oracles.

Now consider `Send` queries sent to users. Queries to initiate a protocol run are trivially simulated. Queries that include ciphertexts must be answered by either accepting or rejecting depending on whether the ciphertext is valid. Because `forge` does not occur we know that \mathcal{A} cannot form a valid ciphertext unless it was output as a result of a `Send` query to the server. Therefore \mathcal{X} has seen every valid ciphertext before and can respond with acceptance when these are seen. Ciphertexts that \mathcal{X} has not seen are rejected.

- Reveal**(U, s) \mathcal{X} knows all session keys that have been accepted, with the possible exception of the one that has been injected in C_b and C'_b . If \mathcal{A} asks for the key for this special case then \mathcal{X} aborts with failure. Otherwise \mathcal{X} can return the correct key.
- Corrupt**(U, K) \mathcal{X} generated all the long-term keys except for those of U_i and U_j . If either of these two parties is corrupted then \mathcal{X} aborts with failure. Otherwise \mathcal{X} can return the correct long-term key.
- Test**(U, s) Suppose that the two tickets C_b and C'_b were accepted by oracles $\Pi_{U_i}^{s_i}$ and $\Pi_{U_j}^{s_j}$. If $(U, s) \notin \{(U_i, s_i), (U_j, s_j)\}$ then \mathcal{X} halts and fails. Otherwise, \mathcal{X} returns the key or a random value.

Eventually \mathcal{A} halts and outputs a bit b . \mathcal{X} returns that same bit to the challenger.

This completes the description of \mathcal{X} . Let *lucky* be the event that \mathcal{X} does not fail during the **Test** query. When *lucky* occurs, \mathcal{X} wins game \mathcal{G}'_1 whenever \mathcal{A} is successful. This means that $\Pr(\text{success}_{\mathcal{X}} | \text{lucky}) \geq \Pr(\text{success}_{\mathcal{A}} | \overline{\text{forge}})$. We also have $\Pr(\text{lucky}) \geq 1/(Q^2 \cdot Q_S)$. Putting these together we obtain:

$$\Pr(\text{success}_{\mathcal{A}} | \overline{\text{forge}}) \leq Q^2 \cdot Q_S \cdot Q_H \cdot \Pr(\text{success}_{\mathcal{X}}). \quad (2)$$

3.2.3 Conclusion of Security Proof We know that N , Q_S , and Q_H are polynomial in the security parameter k and ϵ is negligible by definition. Combining equations 1 and 2 we obtain the following result, which shows that if the authenticated encryption algorithm used in the protocol is secure, then the protocol is also secure.

Theorem 1 *Let \mathcal{A} be any polynomial time adversary against the security of the protocol and \mathcal{H} is modelled as a random oracle. Then there is an integrity adversary, \mathcal{F} , and a confidentiality adversary, \mathcal{X} against the encrypted authentication algorithm such that*

$$\Pr(\text{success}_{\mathcal{A}}) \leq Q \cdot \Pr(\text{success}_{\mathcal{F}}) + Q^2 \cdot Q_S \cdot Q_H \cdot \Pr(\text{success}_{\mathcal{X}}).$$

4 An Extension to the BR93 Model

Despite Theorem 1 being proven in the previous section, Protocol 1 has a significant weakness in a realistic setting (similar to the weakness acknowledged by Boyd in his protocol [8]). It is inevitable that from time to time long-term keys of users will be compromised, e.g., theft of a device containing the key. It seems natural that in such a case the user should be

re-issued with a new long-term private key and then allowed to continue using the protocol. For many server-based protocols this procedure will not influence the protocol security. However, for Protocol 1 this is not the case. It is easy to see that an adversary who obtains a long-term key of a user can continue to use it to masquerade as that user even after a new long-term key has been issued. The reason that this attack is possible even though we have proven the protocol secure, is that there is no notion of replacing a long-term key in the BR93 model: once a party has been corrupted it must remain so. In other words, once a party, say U_1 , is corrupted and its long-term key revealed to the adversary, \mathcal{A} , U_1 is no longer considered fresh in the sense of Definition 2.

One of the motivations for this work is to remove a known weakness of the protocol of Boyd [8] under the effect of a compromise of a long-term key. That is, even if the adversary, \mathcal{A} , has corrupted some party, say U_1 , \mathcal{A} should not be able to impersonate U_1 using the compromised long-term key (of U_1) after a new long-term key has been issued to U_1 . In order to take into account this sort of attack we add a new query called **Reset** to the list of actions that an adversary is allowed to perform and adjust the definition of freshness.

Reset Query. The $\text{Reset}(U_i, K_{New})$ query captures the notion of replacement for a compromised long-term key of principal U_i with a new randomly distributed key, K_{New} . When a corrupted U_i is being asked such a Reset query,

- player U_i is re-considered fresh in the sense of Definition 2,
- any oracle(s) $\Pi_{U_i^1}, \dots, \Pi_{U_i^{\delta-1}}$ that were activated before the Reset query are unfresh in the sense of Definition 2, and
- subsequent oracles $\Pi_{U_i^\delta}, \Pi_{U_i^{\delta+1}}, \dots$ are considered fresh in the sense of Definition 2 (unless U_1 is corrupted again).

An adversary, \mathcal{A} who has access to this new query can always defeat Protocol 1 as follows.

1. \mathcal{A} uses Send queries to run the protocol between A and B .
2. Then \mathcal{A} issues a $\text{Corrupt}(A)$ query to obtain the long-term key of A . This enables \mathcal{A} to decrypt the ticket $\{A, B, K_S\}_{K_{AS}}$ sent to A during a previous protocol run with B , and hence obtain the key K_S contained in it.
3. \mathcal{A} now resets A and masquerades as S , replaying the ticket originally sent to B together with any random value for N_A . This activates a

fresh oracle \mathcal{H}_B^s , that will choose a nonce N_B and accept the session key $\mathcal{H}(K_S \parallel N_A \parallel N_B)$.

4. Consequently, \mathcal{A} knows the value of this accepted key, in violation of Definition 3.

In order to avoid the problem, one method is to introduce a validity period for tickets and to issue a blacklist for tickets that have been compromised. This is the method suggested by Crispo, Popescu, and Tanenbaum [10] whereby they show that a large number of users can be accommodated in a practical system. It is easily checked that this prevents the above attack, since revoked tickets cannot be replayed by the adversary. However, such an approach entails a considerable infrastructure (not unlike a public key infrastructure) and might not scale well to a more realistic environment with a large number of participating entities.

5 An Efficient and Provably-Secure Protocol in the Extended Model

5.1 An Efficient Protocol

Protocol 2 describes our proposed key agreement protocol. In Protocol 2, \mathcal{H}_0 and \mathcal{H}_1 are modelled as random oracles, $[\cdot]_{MK}$ denotes the computation of some MAC digest using MAC key, MK , $\{\cdot\}_{K_{US}}$ denotes the encryption of some message using encryption key, K_{US} , that is being shared by some user and the server, and \parallel denotes the concatenation of messages. We assume that \mathbb{G} , q , g , \mathcal{H}_0 , \mathcal{H}_1 are fixed in advance and known to the entire network, and that each party P_i has a long-term symmetric key, $K_{P_i,S}$, shared with the server, S .

Informally, Protocol 2 removes the known weakness of Protocol 1, as described below.

1. Upon completion of an execution of Protocol 2, A and B have accepted session keys of the same value, $K_{AB} = \mathcal{H}_0(A \parallel B \parallel g^{N_A} \parallel g^{N_B} \parallel g^{N_B N_A})$.
2. Suppose the adversary, \mathcal{A} , compromises the long-term key of A , K_{AS} . With knowledge of K_{AS} , \mathcal{A} can decrypt $\{A, B, g^{N_A}\}_{K_{AS}}$ and learn g^{N_A} . \mathcal{A} also knows g^{N_B} from observing the Protocol 2's execution. However, finding $g^{N_B N_A}$ is equivalent to solving the CDH problem (recall that N_A has been deleted from the internal state of A upon completion of the execution of Protocol 2). Moreover, this implies that Protocol 2 provides *forward secrecy* since the knowledge of the compromised long-term keys, K_{AS} or K_{BS} , does not allow the adversary to find the session key, $K_{AB} = \mathcal{H}_0(A \parallel B \parallel g^{N_A} \parallel g^{N_B} \parallel g^{N_B N_A})$.

| A | S | B |
|---|---|---|
| $N_A \in_R \{0, 1\}^k$ | $\{A, B, g^{N_A}\}_{K_{AS}} \xrightarrow{\hspace{1cm}}$ | $\{A, B, g^{N_A}\}_{K_{BS}}$ |
| | | $N_B \in_R \{0, 1\}^k; SID_B = g^{N_A} g^{N_B}$ $MK_{AB} = \mathcal{H}_1(A B SID_B (g^{N_A})^{N_B})$ $K_{AB} = \mathcal{H}_0(A B SID_B (g^{N_A})^{N_B})$ |
| $SID_A = g^{N_A} g^{N_B}$ $MK_{AB} = \mathcal{H}_1(A B SID_A (g^{N_B})^{N_A})$ Verify received MAC digest, $["1", B, A, SID_B]_{MK_{AB}}$ $K_{AB} = \mathcal{H}_0(A B SID_A (g^{N_B})^{N_A})$ | $g^{N_B}, ["1", B, A, SID_B]_{MK_{AB}} \xleftarrow{\hspace{1cm}}$ | Delete N_B |
| Delete N_A Status: ACCEPTED | $["2", A, B, SID_A]_{MK_{AB}} \xrightarrow{\hspace{1cm}}$ | Verify $["2", A, B, SID_A]_{MK_{AB}}$ Status: ACCEPTED |

Protocol 2: A new key agreement protocol with key confirmation and forward secrecy

5.2 Security Proof

Theorem 2 *Assuming the Computational Diffie-Hellman (CDH) assumption is satisfied in \mathbb{G} , Protocol 2 is a secure key agreement protocol providing key confirmation and forward secrecy when \mathcal{H}_0 and \mathcal{H}_1 are modeled as random oracles and if the underlying message authentication scheme and encryption scheme are secure in the sense of existential unforgeability under adaptive chosen-message attack and indistinguishable under chosen-plaintext attack respectively.*

The validity of Protocol 2 is straightforward to verify and we concentrate on the indistinguishability requirement. The security is proved by finding a reduction to the security of the underlying message authentication scheme and the underlying encryption scheme. The security of Protocol 2 is based on the CDH problem in the random oracle model. An adversary, \mathcal{A} , can get information about a particular session key $K_{ij} = \mathcal{H}_0(i || j || SID_i^k || g^{N_i N_j})$ if \mathcal{A} has queried the random oracle on the point $i || j || SID_i^k || g^{N_i N_j}$. This will allow us to solve the CDH problem with probability related to that of \mathcal{A} 's success probability.

The general notion follows that of the proof presented in Section 3.2. For consistency, let Q be the upper bound of the number of parties in \mathcal{G} , Q_S be the maximum number of Send queries that \mathcal{A} will ask of the server, and Q_H be the maximum number of hash queries that \mathcal{A} will ask of the server. The proof is divided into two parts since the adversary, \mathcal{A} , can

either gain her advantage against Protocol 2 while forging a MAC digest or gain her advantage against Protocol 2 without forging a MAC digest.

The proof then concludes by observing that $\text{Adv}^{\mathcal{A}}(k)$ is negligible when \mathcal{H}_0 , and \mathcal{H}_1 are modeled as random oracles and if the underlying message authentication scheme and encryption scheme are secure in the sense of existential unforgeability under adaptive chosen-message attack and indistinguishable under chosen-plaintext attack respectively, and therefore Protocol 2 is also secure.

5.2.1 Integrity Breaker We assume that there exists an adversary, \mathcal{A} , against Protocol 2. We then construct an integrity breaker, \mathcal{F}_1 , that make use of \mathcal{A} to break the underlying message authentication scheme. Let **Forge** be the event that, for some instance Π_i^k with partner P_j , the adversary queries $\text{Send}(i, k, m)$ and (1) neither P_i nor P_j were ever corrupted; (2) m was never sent by P_j and (3) Π_i^k computes a valid session key.

Lemma 4. *There is an efficient algorithm \mathcal{F}_1 defined using \mathcal{A} such that if **Forge** occurs with non-negligible probability then $\text{success}_{\mathcal{F}_1}$ occurs with non-negligible probability.*

Similar to the proof presented in Section 3.2, the integrity breaker, \mathcal{F}_1 , is able to simulate the view of \mathcal{A} and answers all the oracle queries of \mathcal{A} . \mathcal{F}_1 will continue the simulation until either the event that **Forge** occurs or until \mathcal{A} halts.

Let $\Pr(\text{Forge}_{i,j})$ be the probability that **Forge** occurs for a specific pair of parties i, j . Clearly, we have $\Pr(\text{Forge}) \leq N^2 \cdot \Pr(\text{Forge}_{i,j})$ since we can embed an instance of the CDH problem within the one-time MAC key. Now, if we replace the key, $K_{ij} = g^{U_1 U_2}$ by a random element from \mathbb{G} , this does not affect $\Pr(\text{Forge}_{i,j})$ by more than a factor loss of ϵ . However, the probability that $\Pr(\text{Forge}_{i,j})$ occurs when K_{ij} is truly random is at most ϵ' by the security of the MAC. Therefore, we arrive at the upper bound $\Pr(\text{Forge}) \leq N^2(\epsilon + \epsilon')$.

All queries by the adversary, \mathcal{A} , can be answered normally by \mathcal{F}_1 . \mathcal{F}_1 continues the simulation until $\Pr(\text{Forge})$ happens or until \mathcal{A} terminates. Thus, the simulation is perfect so long \mathcal{F}_1 does not terminate. We then arrive at the following upper bound.

$$\Pr(\text{Forge}) \leq Q \cdot N^2(\epsilon + \epsilon') \cdot \Pr(\text{success}_{\mathcal{F}_1}). \quad (3)$$

5.2.2 Confidentiality Breaker We construct a confidentiality breaker, \mathcal{X}_1 , using \mathcal{A} , as shown in the attack game, $\mathcal{G}_{\mathcal{X}_1}$. The idea underlying this proof follows that presented in Section 3.2.2. \mathcal{X}_1 is given access to the encryption oracles associated with the two random keys K and K' . \mathcal{X}_1 then generates the long-term key for all users, except two randomly selected users, U_i and U_j .

Let $\Pr(\text{Success}_{\mathcal{A}}|\overline{\text{Forge}}_{i,j})$ be the probability that the adversary, \mathcal{A} , manages to gain an advantage without forging a MAC digest for a specific pair of parties i, j . Clearly, we have $\Pr(\text{Success}_{\mathcal{A}}) \leq N^2 \cdot \Pr(\text{Success}_{\mathcal{A}}|\overline{\text{Forge}}_{i,j})$ since we can embed an instance of the CDH problem. Now, if we replace the key, $K_{ij} = g^{U_1 U_2}$ by a random element from \mathbb{G} , this does not affect $\Pr(\text{Forge}_{i,j})$ by more than a factor loss of ϵ . However, the probability that $\Pr(\text{Forge}_{i,j})$ occurs when K_{ij} is truly random is at most ϵ' by the security of the MAC. Therefore, we arrive at the upper bound $\Pr(\text{Forge}) \leq N^2(\epsilon + \epsilon')$.

\mathcal{X}_1 runs \mathcal{A} and answers all oracle queries from \mathcal{A} , as follows:

- For queries $\mathcal{H}_0(i||j||SID_i^k||Z)$, if this query was asked before, then return the previous answer. Otherwise, return a random value, $v \in_R \{0, 1\}^k$. In addition, store this answer together with the query in a list of DH-tuples.
- For Send, Reveal, Corrupt, and Test queries, \mathcal{X}_1 answer them honestly.
- In the event that a Reset(U) query is asked, \mathcal{X}_1 is also able to answer this correctly by returning a new random long-term key generated using the key generation algorithm \mathcal{K}_k . \mathcal{X}_1 has to maintain a table containing the internal state associated with U .
 1. If U has been corrupted, then the status of U is updated to be fresh. Otherwise, \mathcal{X}_1 halts the simulation.
 2. Then any oracle(s) $U^1, \dots, U_i^{\delta-1}$ that were activated before the Reset query are considered unfresh, while subsequent oracles $U_i^\delta, U_i^{\delta+1}, \dots$ are considered fresh.
- At the conclusion of the game simulation if \mathcal{X}_1 has not terminated, \mathcal{X}_1 randomly chooses a tuple, (U_a, U_b, U_{ab}) , from its list of DH-tuples, finds a and b such that $U_a = U_1 g^a$ and $U_b = U_2 g^b$, and outputs $\frac{U_{ab}}{U_2^a U_1^b g^{ab}}$.

We let

- event1 be the event that, for some $i, j \in [q_p]$, \mathcal{A} at some point queries the random oracle at a point $i||j||SID_i^k||g^w$, whereby both P_i and P_j are fresh (i.e., not corrupted) in the entire course of \mathcal{G} , and $W = g^{N_i N_j}$.

- **event2** be the event that, for some $i, j \in [q_p]$, \mathcal{A} at some point queries the random oracle at a point $i||j||SID_i^k||g^x$ for some k , $SID_i^k = U||V = g^{N_i}g^{N_j}$ and $X = g^{N_iN_j}$.

The probability that \mathcal{X}_1 returns the correct answer is at least $\frac{Pr_{\mathcal{A}}[\overline{\text{event1} \wedge \text{corrupted}}]}{Q_H \cdot N^2 \cdot \epsilon}$, since the simulation of $\mathcal{G}_{\mathcal{X}_1}$ is perfect until the point, if any, that **event1** or **event2** occurs. In the event that **event2** occurs, with probability $\frac{1}{Q_H}$, \mathcal{X}_1 selects a tuple, (U_a, U_b, U_{ab}) , from its list of DH-tuples, for which $U_{ab} = g^{\alpha_1 \alpha_2}$ where $\alpha_1 := \log_g U_a$ and $\alpha_2 := \log_g U_b$. In other words, this tuple, (U_a, U_b, U_{ab}) , selected by \mathcal{X}_1 is a DH-tuple. Then \mathcal{X}_1 has outputted a correct solution to the CDH instance. Thus, $Pr_{\mathcal{A}}(\text{event2}) \leq Q_H \cdot \epsilon$.

Since we have shown that both $Pr_{\mathcal{A}}[\overline{\text{event1} \wedge \text{corrupted}}] \leq Q_H \cdot N^2 \cdot \epsilon$ and $Pr_{\mathcal{A}}[\text{event2}] \leq Q_H \cdot \epsilon$, we have

$$\Pr(\text{success}_{\mathcal{A}}|\overline{\text{Forge}}) \leq (Q_H \cdot N^2 \cdot \epsilon + Q_H \cdot \epsilon) \cdot \Pr(\text{success}_{\mathcal{X}_1}). \quad (4)$$

5.2.3 Conclusion of Security Proof We know that N , Q_S , and Q_H are polynomial in the security parameter k and ϵ is negligible by definition. Therefore, by combining equations 3 and 4 we conclude the proof of Theorem 2.

6 Comparative Security and Efficiency

Similar to the work of Gong [14] and Boyd [8], our motivation is to design protocols efficient in both messages and rounds. Therefore, we present a comparative summary of Protocols 1 and 2 with other similar server-based key establishment protocols of Gong [13, 14] as described in Table 1. In particular, we compare Protocols 1 and 2 with the protocol classes defined by Gong where both users contribute to the session key.

In terms of both messages and rounds, we observe that

- Protocol 1 is as efficient as that obtained by Gong [14] for server-based protocols with similar goals using timestamps.
- Protocol 2, which provides key confirmation, breaks Gong’s lower bound since an extra round is required for providing key confirmation in the first three protocols described in described in Table 1.

Moreover, Protocol 2 removes the known weakness of Protocol 1 under the effect of a compromise of a long-term key as described in Section 5.1 at the expense of computational overhead (i.e., Protocol 2 is more computational expensive due to the use of Diffie–Hellman exponentiation).

| Protocols | Messages | Security proof? |
|--|---------------|--|
| The following three protocols do not provide key confirmation (KC). However, key confirmation can be provided at the cost of an extra message. | | |
| 1. Protocol 1 | 3 (+1 for KC) | Proven secure in the BR93 model. |
| 2. Timestamp-based protocol [14] | 4 (+1 for KC) | No. |
| 3. Nonce-based protocol [14] | 5 (+1 for KC) | No. |
| The following three protocols provide key confirmation. | | |
| 4. Alternative protocol using uncertified keys [14] | 5 | No. |
| 5. Hybrid protocol [13] | 5 | No. |
| 6. Protocol 2 | 4 | Proven secure in the extended BR93 model. Protocols proven secure in the extended BR93 model will also be secure in the BR93 model. Moreover, Protocol 2 provides both key confirmation and forward secrecy. |

Table 1. A comparative summary

We also remark that another attractive feature of Protocol 2 is the extension which allows session keys to be renewed in subsequent sessions without the server's further involvement. The extension to Protocol 2 that allows the session key to be renewed is described in Protocol 3. This entails A and B exchanging new nonces N'_A and N'_B and computing the new session key as $K'_{AB} = \mathcal{H}_1(A||B||S||N'_A||N'_B||g^{N_A N_B}) = K'_{BA}$.

| | | |
|-------------------------|---|-------------------------|
| A | | B |
| $N'_A \in_R \{0, 1\}^k$ | $\xrightarrow{A, N'_A} \xleftarrow{B, N'_B}$ | $N'_B \in_R \{0, 1\}^k$ |
| | $SID_{A'} = (N'_A N'_B) = SID_{B'}$ | |
| | $K'_{AB} = \mathcal{H}_1(A B S SID_{A'} g^{N_A N_B}) = \mathcal{H}_1(A B S SID_{B'} g^{N_A N_B}) = K'_{BA}$ | |

Protocol 3: An extension to Protocol 2

Recall from our earlier discussion in Section 5.1 that exposing the long-term key will not enable the adversary to learn the CDH key, $g^{N_A N_B}$. Neither will the adversary be able to learn the CDH key, $g^{N_A N_B}$, by ex-

posing an earlier agreed session key, K_{AB} . If the adversary is able to learn $g^{N_A N_B}$ by exposing K_{AB} , then we will be able to make use of such an adversary to break the underlying CDH problem. Therefore, the extension presented in Protocol 3 is still possible even if the long-term key or the earlier session key have been compromised. However, this is not the case for the other server-based three-party key establishment protocols described in Table 1.

7 Conclusions

We proved the security of another protocol example, revised protocol of Boyd [8] – Protocol 1, in the BR93 model. Although Protocol 1 is known to be insecure under reasonable assumptions, this does not show up in the original BR93 model because there is no capability for the adversary to reset corrupted principals. We then extended the BR93 model so that it allows more realistic adversary capabilities, which allows us to detect a known weakness of Protocol 1 that cannot be captured in the original (BR93) model. We then presented another protocol (i.e., Protocol 2) that is efficient in both messages and rounds, and then proved Protocol 2 secure in the extended BR93 model and the random oracle model.

Future Work. This work allows us to detect a known weakness of the Boyd key agreement protocol [8] that cannot be captured in the original BR93 model. It would be interesting to know what other (symmetric-key) protocols may also have this property. Another possible extension is to investigate and propose a modular proof approach with a formal statement of security that allows server-based three-party key establishment protocols like those introduced in Table 1 to renew session key(s) in subsequent sessions without the server’s further involvement, even in the event that the long-term key or the earlier session key are compromised.

References

1. M. Bellare, A. Boldyreva, and S. Micali. Public-key Encryption in a Multi-User Setting: Security Proofs and Improvements. In *EUROCRYPT 2000*, volume 1807/2000 of *LNCS*, pages 259 – 274. Springer-Verlag, 2000.
2. M. Bellare and C. Namprempe. Authenticated Encryption: Relations Among Notions and Analysis of the Generic Composition Paradigm. In *ASIACRYPT 2000*, volume 1976/2000 of *LNCS*, pages 531–545. Springer-Verlag, 2000.
3. M. Bellare, D. Pointcheval, and P. Rogaway. Authenticated Key Exchange Secure Against Dictionary Attacks. In *EUROCRYPT 2000*, volume 1807/2000 of *LNCS*, pages 139 – 155. Springer-Verlag, 2000.

4. M. Bellare and P. Rogaway. Entity Authentication and Key Distribution. In *CRYPTO 1993*, volume 773/1993 of *LNCS*, pages 110–125. Springer-Verlag, 1993.
5. M. Bellare and P. Rogaway. Random Oracles Are Practical: A Paradigm For Designing Efficient Protocols. In *ACM CCS 1993*, pages 62–73. ACM Press, 1993.
6. M. Bellare and P. Rogaway. Provably Secure Session Key Distribution: The Three Party Case. In *ACM STOC 1995*, pages 57–66. ACM Press, 1995.
7. J. Black. The Ideal-Cipher Model, Revisited: An Uninstantiable Blockcipher-Based Hash Function (Extended version available from <http://eprint.iacr.org/2005/210>). In *EUROCRYPT 2006*, LNCS. Springer-Verlag, 2006. To Appear.
8. C. Boyd. A Class of Flexible and Efficient Key Management Protocols. In *CSFW 1996*, pages 2–8. IEEE Computer Society Press, 1996.
9. D. Catalano, D. Pointcheval, and T. Pornin. Trapdoor Hard-to-Invert Group Isomorphisms and Their Application to Password-based Authentication. *Journal of Cryptology*, 2006. To Appear.
10. B. Crispo, B. C. Popescu, and A. S. Tanenbaum. Symmetric Key Authentication Services Revisited. In *ACISP 2004*, volume 3108/2004 of *LNCS*, pages 248–261. Springer-Verlag, 2004.
11. C. Gentry, P. MacKenzie, and Z. Ramzan. Password Authenticated Key Exchange Using Hidden Smooth Subgroups. In *ACM CCS 2005*, pages 299–309. ACM Press, 2005.
12. O. Goldreich and Y. Lindell. Session-Key Generation using Human Passwords Only (Updated Version available from <http://eprint.iacr.org/2000/057/>). In *CRYPTO 2001*, volume 2139/2001 of *LNCS*, pages 408–432. Springer-Verlag, 2001.
13. L. Gong. Using One-Way Functions for Authentication. *ACM SIGCOMM Computer Communications Review*, 8(11):8–11, 1989.
14. L. Gong. Lower Bounds on Messages and Rounds for Network Authentication Protocols. In *ACM CCS 1993*, pages 26–37. ACM Press, 1993.