Analysis of Authenticated Key Exchange under Bad Randomness

Authors: Nabiha Asghar & Yin Zhang

1 Introduction

In cryptography, a Key Exchange Protocol allows two parties to establish a common encryption/ decryption key (called a 'session key') so that they can communicate over an insecure channel in encrypted mode. An Authentication Protocol allows them to confirm each other's identity in a secure manner. A hybrid of the two is an Authenticated Key Exchange (AKE) protocol. It consists of two probabilistic polynomial time (PPT) algorithms that achieve the two goals of authentication and key exchange simultaneously. A key feature of an AKE protocol is its use of random coins (i.e. numbers where each bit is a random flip of a coin) to achieve its goals. For security purposes, these coins are expected to be truly random. Therefore, it is only natural to consider a scenario where the adversarial power encompasses the ability to meddle with the randomness of these coins and consequently compromise the security of the protocol as well as the integrity of the long-lived secret keys of the participants. This project is a study of the security of AKE protocols, given this adversarial power.

2 Motivation

Typically in an AKE protocol, a pseudo-random number generator (PRNG) is used to generate the random coins. A PRNG is a deterministic algorithm that requires a seed for its initialization. It then outputs 'pseudo-random' coins that are computationally indistinguishable from truly random coins. Due to this indistinguishability property, pseudo-random coins are considered safe to be used in an AKE protocol. The only requirement is that the seed of the PRNG should be fresh (i.e. not repeated) and truly random [7] because an adversary, who knows the PRNG algorithm and the seed used to initialize a particular run of the PRNG, can predict the output of the PRNG and compromise the security of the protocol. Hence the goal boils down to finding a source of truly random seeds, and then protecting it against a security breach.

As the first step, let us consider a few examples of sources for seeds and show how some of them may be unsafe.

i) One may consider using a computer deterministically to generate random seeds. This defies the definition of 'random'. In general, turing machines are incapable of generating truly random numbers.

ii) The current time on a computer's real-time clock, accessible by the time(0) command in C++ for example, is a commonly used seeding option. This is not entirely safe because it is possible

to track the approximate sending time of a data packet over a network. This shrinks the domain considerably, allowing an adversary to use brute force to obtain the exact clock time used as the seed.

iii) Lately, the idea of tapping the truly unpredictable randomness of the chaotic universe around us is gaining popularity. This involves the use of physical/natural events as triggers/seeds for the PRNG. One example is network traffic, or a user's mouse clicks and keyboard strokes [10, 15]. Another example is the servers of http://www.random.org that use radio receivers to pick up atmospheric noise caused by rain, thunderstorms etc. Another source of truly random seeds is HotBits[1], which makes use of the 'inherent uncertainty in the quantum mechanical laws of nature'. The random seeds are generated by timing successive pairs of radioactive decays detected by a Geiger-Müller tube connected to a computer.

iv) Hardware is potentially a good source for random seeds. Intel's Bull Mountain[2] technology is an example of a system that uses thermal noise within the resistors of a CPU to generate truly random seeds.

We can see that there is no dearth of sources of truly random seeds. The question is whether these seeds can be protected after they have been generated. More specifically, are there any practical situations where the adversary has direct or indirect access to the entropy pool [3] of seeds? The answer is in affirmative. If the adversary has physical access to a hardware source or if he can control the events of a non-hardware source that is being used to generate seeds, he can manipulate the data in the entropy pool. He may also gain access to the entropy pool itself. In a Linux operating system, for example, the entropy may be stored in a buffer. An adversary may gain access to this buffer if it is not well-protected. There may also be situations, such as in Virtual Machines [17] or smart cards [4], where an adversary can reset a machine to make an AKE protocol reuse some random coins across different sessions. An example of this may occur when an administrator takes snapshots of the current system state of a Virtual Machine (VM) from time to time as regular backups, so that if it crashes, it can be reverted back to a previous good state using the snapshots. To perform an adversarial reset, an adversary makes a VM crash, perhaps by a Denial of Service (DoS) attack. This causes the system administrator to revert the VM to a previous good state, thereby making it reuse some of the older random coins.

Therefore, it is imperative to undertake a study of how to avoid the case of 'bad randomness', i.e. a situation where the randomness generated within an AKE protocol has been compromised by an adversary who can

1) directly pick random coins for the AKE participants (called the 'Reset-1 Attack'), or
2) reset a participant to make it reuse some random coins (called the 'Reset-2 Attack').

---

[1] http://www.fourmilab.ch/hotbits/

[2] http://spectrum.ieee.org/computing/hardware/behind-intels-new-randomnumber-generator/0

[3] Entropy is a collection of random bits collected by an operating system. Typically, these random bits come from hardware sources such as mouse movements.

3 Project Goals and Layout

This project is a self-contained discourse on the security of AKE protocols under Reset-1 and Reset-2 attacks. Our goal is to study in detail the work of Yang et al. [18] and to present their analysis in an elaborate and rigorous form by filling in any logical, computational or analytical details that the authors may have left out intentionally or otherwise. The specific details of our additional contributions are given at the end of this section.

Section 4 gives formal definitions of various terms used throughout this discourse and defines the formal security models for Reset-1 and Reset-2 attacks. Section 5 gives concrete examples of some famous AKE protocols that are insecure under either of the two attacks. Section 6 describes a generic method to transform any Reset-2 secure AKE protocol to a new one which is also Reset-1 secure. This is shown with the help of two theorems. Section 7 proposes two new protocols and gives formal proofs of their security under the two attacks (via two theorems). Section 8 gives concluding remarks.

Our contribution to the work of Yang et al. [18] is threefold.

1. In section 5, we give Reset-1 and Reset-2 attacks on ISO [1, 5] and SIGMA [6, 13] protocols, which have been completely omitted in the original paper.
2. In Sections 6 and 7, we give the proofs of Theorems 1 to 4 in complete detail, filling in the various gaps in arguments and computations in the original paper. Moreover, the proofs in the original paper are presented in a dense way and are hard to follow. We strive hard to improve the presentation to make everything easily comprehensible to readers.
3. We implement the Reset-2 attack on the HMQV [12] protocol given in section 5. The implementation is done in Maple, using sockets to simulate each party participating in the protocol and the attack.

4 Definitions and Security Models

We begin by giving descriptions of various components of an AKE Protocol and the definition of non-negligibility in Section 4.1. Sections 4.2 and 4.3 give the complete security models for Reset-1 and Reset-2 attacks respectively. Section 4.4 gives the security models of the stronger version of the attacks. For the sake of consistency and uniformity between this essay and the original paper, we keep these definitions and descriptions very similar to those given by Yang et al. [18].

4.1 AKE Protocol Descriptions & the Notion of Non-negligibility

An **Authenticated Key Exchange (AKE) protocol** consists of two probabilistic polynomial time algorithms: 1) a Secret Key Generation algorithm called SKG, which returns a public key and a secret key upon each invocation, and 2) a protocol execution algorithm P that maps strings to strings. The set of **Protocol Participants** is denoted by $\mathcal{U} \cup \mathcal{MU}$, where $\mathcal{U}$ is a non-empty set of users registered in the network in the initialization phase and $\mathcal{MU}$ is a set of malicious parties added into the network by the adversary after the initialization phase. Each user $U \in \mathcal{U}$ has a unique ID string of a fixed length,

denoted by U. Each malicious user $M \in \mathcal{MU}$ also has a distinct, fixed-length ID string denoted by M which has never been used to name another party in the network. Each user $U \in \mathcal{U}$ has a pair of **Long-Lived Keys** (pk_U, sk_U) generated by the SKG, where pk_U is the public key and sk_U is the secret key. However, for each user $M \in \mathcal{MU}$, the adversary can set pk_M to any value that has never been used as the public key of another user in the system; sk_M is then chosen by the adversary according to the value of pk_M.

An **Instance** of a user denotes the readiness of a user to take part in a session. There may be multiple instances of a user running concurrently, so the i'th instance of a user U is denoted by Π_U^i. At the time a new instance of a user U is created, a unique instance number i is chosen for U, a sequence of random coins N_U^i are tossed (i.e. a random number is chosen) and feeded to that instance, and the instance enters the 'ready' state. The **Protocol Execution** Algorithm determines how an instance of a user responds to messages from other users. It is a probabilistic algorithm that maps strings to strings as follows.

$$(M_{out}, acc, term_U^i, sid_U^i, pid_U^i, ssk, St_U^i) \rightarrow P(1^k, U, pk_U, sk_U, St_U^i, M_{in}).$$

More specifically, when the instance Π_U^i receives the message M_{in} from another user or instance, Π_U^i runs the protocol algorithm P with the inputs U, pk_U, sk_U, M_{in}, a number 1^k dependent on the security parameter k, and the state information $St_U^i = (N_U^i$, 'ready'). The output consists of an outgoing message M_{out} from Π_U^i to the sender, the decision acc (accept/reject) made by Π_U^i, and a third component $term_U^i$ which indicates whether the protocol execution has been terminated. Optionally, a session ID sid_U^i and a partner ID pid_U^i may be generated as an output, where the partner ID refers to the ID of the user who sent M_{in} to Π_U^i. When $acc = accept$, ssk is another output of the protocol execution. This ssk is the common session key shared by Π_U^i and the sender which is to be used by the upper layer applications. We assume that the state information St_U^i is erased from the memory of U when $term_U^i = true$ i.e. when this particular run of the protocol terminates.

Finally, the **Partnership** between two instances is defined via partner ID (pid) and session ID (sid). As described above, pid is the ID of the user with which the instance Π_U^i believes it has just exchanged a key, and sid is an ID which uniqely labels the AKE session. Two instances Π_U^i and Π_V^j are partners if $pid_U^i = V$, $pid_V^j = U$ and $sid_U^i = sid_V^j$.

The Notion of Non-Negligibility: Since we are in the paradigm of randomized algorithms, we make extensive use of the notion of non-negligibility. We say that a function $f : \mathbb{N} \rightarrow \mathbb{R}$ is negligible if for any positive integer c, there exists an integer N_c such that for all integers $n > N_c$, $|f(n)| < \frac{1}{n^c}$. Equivalently, a function $f : \mathbb{N} \rightarrow \mathbb{R}$ is non-negligible if there exists a positive integer c such that for any integer N_c, there exists an integer $n > N_c$ such that $|f(n)| \geq \frac{1}{n^c}$.

4.2 Security Model for Reset-1 Attack

In the Reset-1 model, we consider the scenario where the randomness of each instance is completely controlled by the adversary. In other words, the adversary can choose the random coins for each instance. The formal definition of the Reset-1 model is given in terms of the game RAKE-1, described

in Figure 1. Simply put, the goal of the adversary A is to distinguish between some random number and the session key ssk_{U*}^{i*} of a particular target instance Π_{U*}^{i*}. A plays the game RAKE-1, which allows him to make six different types of queries. At the end, the game outputs a number which is equally likely to be either a random key in the session-key space or the session key of Π_{U*}^{i*}. If A is able to correctly distinguish between the two with high probability, he wins the game and we say that the AKE protocol is insecure in the Reset-1 model.

Note that the session key is typically a function of the long-lived secret key and the random coins. In the Reset-1 model, the adversary knows the random coins. If he also learns one of the two users' long-lived secret key, he can trivially compute the session key. We therefore assume in this model that the long-lived keys of all the users in $\mathcal{U}$ are securely generated using fresh random coins that are not known to the adversary. We now describe each of the eight procedures given in Figure 1. Six of them $(2-7)$ are the oracle queries that capture different adversarial capabilities.

procedure Initialize
For all $U \in \mathcal{U}$
  $(\mathrm{pk}_U, \mathrm{sk}_U) \leftarrow_{\$} \mathsf{SKG}(1^k, U)$; $T_U \leftarrow \emptyset$
Timer $\leftarrow 0$; $b \leftarrow_{\$} \{0,1\}$; $\mathcal{MU} \leftarrow \emptyset$
return $\{\mathrm{pk}_U\}_{U \in \mathcal{U}}$

procedure Register(U, pk)
If $(U \in (\mathcal{U} \cup \mathcal{MU}) \vee \mathrm{pk} \in \{\mathrm{pk}_V\}_{V \in \mathcal{U} \cup \mathcal{MU}})$ then
  return *Invalid*
$\mathcal{MU} \leftarrow \mathcal{MU} \cup \{U\}$
return true

procedure NewInstance(U, i, N)
If $(U \notin \mathcal{U} \vee i \in T_U)$ then return *Invalid*
$T_U \leftarrow T_U \cup \{i\}$; $\mathrm{N}_U^i \leftarrow \mathrm{N}$; $St_U^i \leftarrow (\mathrm{N}_U^i, \mathsf{ready})$
$\mathsf{acc}_U^i \leftarrow \mathsf{false}$; $\mathsf{term}_U^i \leftarrow \mathsf{false}$
$\mathsf{sid}_U^i \leftarrow \perp$; $\mathsf{pid}_U^i \leftarrow \perp$; $\mathrm{ssk}_U^i \leftarrow \perp$
return true

procedure Send(U, i, M_{in})
If $(U \notin \mathcal{U} \vee i \notin T_U \vee \mathsf{term}_U^i)$ then return *Invalid*
$(M_{\mathrm{out}}, \mathsf{acc}, \mathsf{term}_U^i, \mathsf{sid}_U^i, \mathsf{pid}_U^i, \mathrm{ssk}, St_U^i)$
  $\leftarrow \mathrm{P}(1^k, U, \mathrm{pk}_U, \mathrm{sk}_U, St_U^i, M_{\mathrm{in}})$
If $(\mathsf{acc} \wedge$ not $\mathsf{acc}_U^i)$ then
  $\mathrm{ssk}_U^i \leftarrow \mathrm{ssk}$; $\mathsf{acc}_U^i \leftarrow \mathsf{true}$
return $(M_{\mathrm{out}}, \mathsf{acc}, \mathsf{term}_U^i, \mathsf{sid}_U^i, \mathsf{pid}_U^i)$

procedure Reveal(U, i)
If $(U \notin \mathcal{U} \vee i \notin T_U)$ then return *Invalid*
Timer $\leftarrow$ Timer $+ 1$; $\mathsf{Time}[\mathbf{Reveal}, (U, i)] \leftarrow$ Timer
return ssk_U^i

procedure Corrupt(U)
If $U \notin \mathcal{U}$ then return *Invalid*
Timer $\leftarrow$ Timer $+ 1$
$\mathsf{Time}[\mathbf{Corrupt}, U] \leftarrow$ Timer
return sk_U

procedure Test(U^*, i^*)
If $U^* \notin \mathcal{U}$ then return *Invalid*
If (not $\mathsf{acc}_{U^*}^{i^*}$) then return *Invalid*
$K \leftarrow_{\$}$ KeySpace
If $b = 0$ then return K
Else return $\mathrm{ssk}_{U^*}^{i^*}$

procedure Finalize(b')
$V^* \leftarrow \mathsf{pid}_{U^*}^{i^*}$
If $V^* \notin \mathcal{U}$ then return false
If $(\mathsf{Time}[\mathbf{Corrupt}, U^*]$
  $\vee \mathsf{Time}[\mathbf{Corrupt}, V^*])$
  return false
If $(\mathsf{Time}[\mathbf{Reveal}, (U^*, i^*)])$
  return false
If $(\exists i, i \neq i^* \wedge \mathrm{N}_{U^*}^i = \mathrm{N}_{U^*}^{i^*})$
  return false
If $(\exists j^* \in T_{V^*}, \mathsf{pid}_{V^*}^{j^*} = U^*$
  $\wedge \mathsf{sid}_{V^*}^{j^*} = \mathsf{sid}_{U^*}^{i^*})$
  If $(\exists j, j \neq j^* \wedge \mathrm{N}_{V^*}^j = \mathrm{N}_{V^*}^{j^*})$
    return false
  If $(\mathsf{Time}[\mathbf{Reveal}, (V^*, j^*)])$
    return false
return $(b = b')$

Figure 1: Game RAKE-1 (Screenshot from Page 5 [18]

1. **Initialize:** In the initialization phase, the SKG algorithm is run and a public key/secret key pair is chosen for each user U. T_U denotes the set of instances of U and is initially empty. A 'Timer' bit is set to 0. A coin 'b' is flipped and its value is saved. This coin/bit is used later in the Test query. The set $\mathcal{MU}$ of malicious users is set to empty. Finally, the public keys of all the users in the network are returned and made public.
2. **Register**(U, pk)**:** This oracle query allows the adversary A to register a new user U with public key pk_U, with the restriction that the ID string U and pk_U must not already exist in the network. U is then added to the set of malicious users.
3. **NewInstance**(U, i, N)**:** This oracle query allows A to initialize a new instance Π_U^i for user U with the binary string N which serves as the sequence of random coins for Π_U^i. The set T_U is updated to include this newly created i'th instance of U. The random coins N_U^i and the readiness of Π_U^i to interact with other instances is captured in the state information St_U^i. Since Π_U^i has not initiated the protocol with any other user or instance yet, acc_U^i and $term_U^i$ are set to false and the session ID, partner ID and session key are set to null.
4. **Send**(U, i, M_{in})**:** This oracle queries allows an adversary to initiate a session with the i'th instance of U (i.e. Π_U^i) by sending it a message M_{in}. Now Π_U^i runs the protocol $P(1^k, U, pk_U, sk_U, St_U^i, M_{in})$ and all the output parameters except the state information and the session key are sent back to A.
5. **Reveal**(U, i)**:** This oracle query allows A to obtain the session key ssk_U^i, for any $U \in \mathcal{U}$ and $i \in T_U$, if Π_U^i has accepted and generated a session key ssk_U^i. The 'Timer' bit which was set to 0 in the initialization phase is updated to 1 and assigned to a variable Time[Reveal, (U, i)], whose non-zero value denotes that the session key of Π_U^i has been revealed to A.
6. **Corrupt**(U)**:** This oracle query allows A to obtain the long-lived secret key sk_U for a user U. The 'Timer' bit is updated and assigned to a variable Time[Corrupt, U] whose non-zero value denotes that U secret key has been corrupted by A.
7. **Test**(U^*, i^*)**:** This oracle query allows A to choose a target instance $\Pi_{U^*}^{i^*}$, where U^* must not be in $\mathcal{MU}$ otherwise A himself is U^*. If $\Pi_{U^*}^{i^*}$ has accepted and is holding a session key $ssk_{U^*}^{i^*}$, then the following happens. If the coin b, flipped in the Initialize phase, is 1, then $ssk_{U^*}^{i^*}$ is returned to A, otherwise a random number is drawn from the session key space and returned to A. This query can only be made once during one game.
8. **Finalize**(b')**:** In the finalization phase, A inputs a bit b' to the game. If $b' = 1$, A believes that the key given to him by the Test procedure is $ssk_{U^*}^{i^*}$, otherwise he believes he has been given some random key. This procedure determines whether A has won the game or not. Note that there are some queries which can help A compute $ssk_{U^*}^{i^*}$ trivially, so we exclude those possibilities in the beginning of this finalization phase.
    i. The first if-condition ensures that the partner V^* of this instance is not A himself.
    ii. The second if-condition ensures that A has not obtained either of the two users' long-lived secret keys sk_{U^*} and sk_{V^*} through the Corrupt query. This is important because A completely controls the randomness in the Reset-1 model, so once a user is corrupted, A can trivially derive all state information and session keys ever generated by that user.
    iii. The third if-condition ensures that A has not obtained $ssk_{U^*}^{i^*}$ through the Reveal query.

iv. The fourth if-condition ensures that A does not obtain ssk_{U*}^{i*} by the *reset-and-replay* attack. In this attack, A first activates a protocol execution between the instances Π_U^i and Π_V^j, where the random coins given to Π_U^i are N_U^i and those given to Π_V^j are N_V^j. Now A activates an instance $\Pi_U^{i'}$ and instead of giving it new random coins, A gives it the same random coins N_U^i. Then he replays messages from Π_V^j, thereby ensuring that $ssk_U^{i'} = ssk_U^i$. Now if A uses the reveal query on Π_U^i, he trivially obtains $ssk_U^{i'}$. This type of attacks imply that as long as the random coins of one instance Π_U^i are being used by another instance $\Pi_U^{i'}$, there is no security guarantee on the session keys generated by these two instances. Therefore, this if-condition ensures that there exists no instance of U^* in the network that is using the same random coins as Π_{U*}^{i*}.

v. The fifth if-condition ensures that if V^* has an instance Π_{V*}^{j*} which is the partner instance of Π_{U*}^{i*}, then A does not obtain the session key ssk_{V*}^{j*}=ssk_{U*}^{i*} by the *reset-and-replay* attack. More specifically, it ensures that ssk_{V*}^{j*} is not using the random coins of any other instance of V^* and its session key has not been obtained by A through the Reveal query.

After ensuring that A did not compute ssk_{U*}^{i*} trivially, this procedure outputs the result of the game, i.e. whether A has won the game by successfully identifying the value of b or not.

We now define the notion of security in the Reset-1 model with respect to game RAKE-1.

Definition 4.2.1: Let $\mathcal{AKE}$ be an AKE protocol. Let A be a Reset-1 adversary against $\mathcal{AKE}$ and n a security parameter. The advantage of A in the game RAKE-1 is defined as

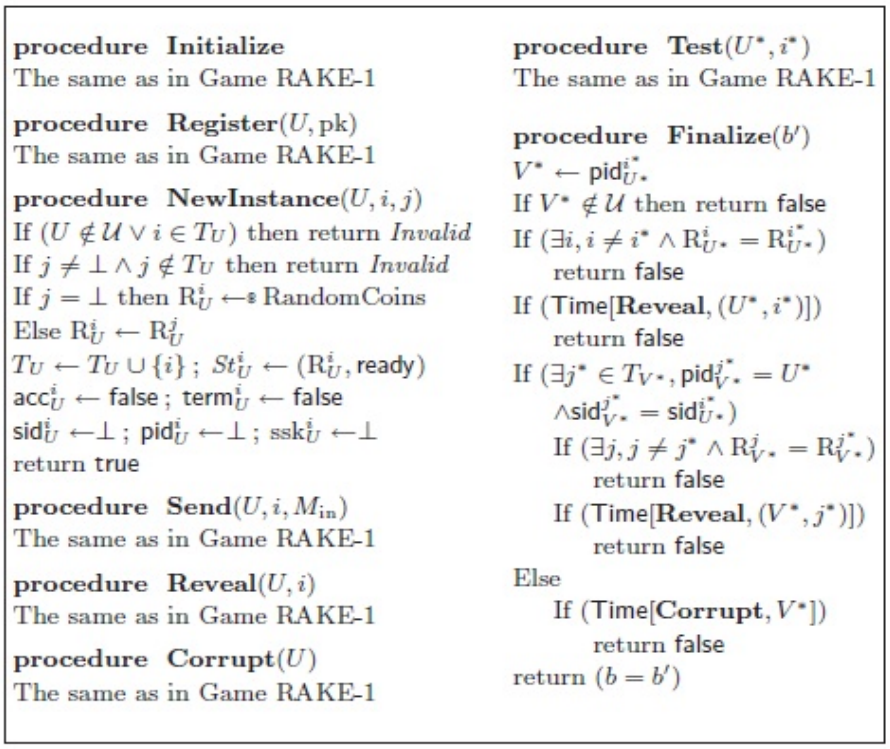
$$\mathbf{Adv}_{\mathcal{AKE},A}^{\text{RAKE-1}}(k) = \Pr[\text{RAKE-1}_{\mathcal{AKE},A}(n) \rightarrow \mathit{true}] - 0.5,$$

where $\Pr[\text{RAKE-1}_{\mathcal{AKE},A}(n) \rightarrow \mathit{true}]$ is the probability of A winning RAKE-1 against $\mathcal{AKE}$. The subtraction of 0.5 is to negate the chances of A winning the game by guessing the answer randomly. We say that $\mathcal{AKE}$ is secure in the Reset-1 model if
i. in the presence of a passive adversary, the two partner instances output the same session key, and
ii. for any probabilistic polynomial time adversary A, $\mathbf{Adv}_{\mathcal{AKE},A}^{\text{RAKE-1}}(n)$ is negligible.

4.3 Security Model for Reset-2 Attack

In the Reset-2 model, we consider the scenario where the adversary cannot set the value of the random coins directly, but he can reset any user to make it reuse some older random coins. The formal definition of the Reset-2 model is given in terms of the game RAKE-2, described in Figure 2, which is very similar to RAKE-1.

The goal of the adversary A in the game RAKE-2 is the same as that in RAKE-1: he needs to distinguish between a random number from the session key space and the session key ssk_{U*}^{i*} of a target instance Π_{U*}^{i*}. This model allows us to define *forward secrecy* , i.e., even if the adversary corrupts the two users and obtains their long-lived secret keys, he should not be able to compute the session key. The definitions of the six procedures **Initialize, Register**(U, pk)**, Send**(U, i, M_{in})**, Reveal**(U, i)**,**



procedure Initialize
The same as in Game RAKE-1

procedure Register(U, pk)
The same as in Game RAKE-1

procedure NewInstance(U, i, j)
If $(U \notin \mathcal{U} \vee i \in T_U)$ then return *Invalid*
If $j \neq \perp \wedge j \notin T_U$ then return *Invalid*
If $j = \perp$ then $\mathrm{R}_U^i \leftarrow_{\$}$ RandomCoins
Else $\mathrm{R}_U^i \leftarrow \mathrm{R}_U^j$
$T_U \leftarrow T_U \cup \{i\}$; $St_U^i \leftarrow (\mathrm{R}_U^i, \mathsf{ready})$
$\mathsf{acc}_U^i \leftarrow \mathsf{false}$; $\mathsf{term}_U^i \leftarrow \mathsf{false}$
$\mathsf{sid}_U^i \leftarrow \perp$; $\mathsf{pid}_U^i \leftarrow \perp$; $\mathsf{ssk}_U^i \leftarrow \perp$
return true

procedure Send(U, i, M_{in})
The same as in Game RAKE-1

procedure Reveal(U, i)
The same as in Game RAKE-1

procedure Corrupt(U)
The same as in Game RAKE-1

procedure Test(U^*, i^*)
The same as in Game RAKE-1

procedure Finalize(b')
$V^* \leftarrow \mathsf{pid}_{U^*}^{i^*}$
If $V^* \notin \mathcal{U}$ then return false
If $(\exists i, i \neq i^* \wedge \mathrm{R}_{U^*}^i = \mathrm{R}_{U^*}^{i^*})$
  return false
If $(\mathsf{Time}[\mathbf{Reveal}, (U^*, i^*)])$
  return false
If $(\exists j^* \in T_{V^*}, \mathsf{pid}_{V^*}^{j^*} = U^*$
  $\wedge \mathsf{sid}_{V^*}^{j^*} = \mathsf{sid}_{U^*}^{i^*})$
  If $(\exists j, j \neq j^* \wedge \mathrm{R}_{V^*}^j = \mathrm{R}_{V^*}^{j^*})$
    return false
  If $(\mathsf{Time}[\mathbf{Reveal}, (V^*, j^*)])$
    return false
Else
  If $(\mathsf{Time}[\mathbf{Corrupt}, V^*])$
    return false
return $(b = b')$

Figure 2: Game RAKE-2 (Screenshot from page 7 [18])

Corrupt(U) and **Test**(U^*, i^*) are exactly the same as those in RAKE-1. The two different procedures and their descriptions are as follows.

1. **NewInstance**(U, i, j)**:** This query allows A to initialize a new instance Π_U^i of user U not by setting its random coins directly, but by making it re-use the random coins R_U^j of the instance Π_U^j that has already been initialized. A can also let $j = \perp$, which means that A can let Π_U^i use fresh random coins unknown to A.
2. **Finalize**(b')**:** In the finalization phase, A inputs a bit b' to the game as in RAKE-1. This procedure determines if A has won the game or not. There are some queries that can help A compute ssk_{U*}^{i*} trivially, so we exclude those possibilities in the beginning of the finalization phase.
   i. The first if-condition ensures that the partner V^* of this instance is not A himself.
   ii. The second if-condition ensures that there exists no instance of U^* in the network that is using the same random coins as Π_{U*}^{i*}, because the existence of such an instance implies that A has already launched a *reset-and-replay* attack against Π_{U*}^{i*} and can trivially compute ssk_{U*}^{i*} as explained in section 4.2.
   iii. The third if-condition ensures that A has not obtained ssk_{U*}^{i*} through the Reveal query.
   iv. The fourth if-condition ensures that if Π_{U*}^{i*} has a partner instance Π_{V*}^{j*}, then A does not use the Reveal query on that instance. Moreover, it ensures that no instance of V^* in the network uses the same random coins as Π_{V*}^{j*}. This is to prevent A from launching a *reset-and-replay* attack against Π_{V*}^{j*}.

v. If $\Pi_{U^*}^{i^*}$ has a partner ID V^* but no partner instance and A corrupts V^*, then an acceptance by $\Pi_{U^*}^{i^*}$ implies that A posed successfully as V^* and thus obtained the session key $ssk_{U^*}^{i^*}$. The last else-condition ensures that this does not happen.

After ensuring that A did not compute $ssk_{U^*}^{i^*}$ trivially, this procedure outputs the result of the game, i.e. whether A has won the game by successfully identifying the value of b or not.

We now define the notion of security in the Reset-2 model with respect to game RAKE-2.

Definition 4.3.1: Let $\mathcal{AKE}$ be an AKE protocol. Let A be a Reset-2 adversary against $\mathcal{AKE}$ and n a security parameter. The advantage of A in the game RAKE-2 is defined as

$$\mathbf{Adv}^{\text{RAKE-2}}_{\mathcal{AKE},A}(n) = \Pr[\text{RAKE-2}_{\mathcal{AKE},A}(n) \rightarrow true] - 0.5,$$

where $\Pr[\text{RAKE-2}_{\mathcal{AKE},A}(n) \rightarrow true]$ is the probability of A winning RAKE-2 against $\mathcal{AKE}$. We say that $\mathcal{AKE}$ is secure in the Reset-2 model if
i. in the presence of a passive adversary, the two partner instances output the same session key, and
ii. for any probabilistic polynomial time adversary A, $\mathbf{Adv}^{\text{RAKE-2}}_{\mathcal{AKE},A}(n)$ is negligible.

4.4 Strong-Corruption Security Models for Reset-1 and Reset-2 Attacks

We may call the models described above the 'weak corruption models', where the adversary can obtain a user's long-lived secret key or any session key through oracle queries. However he cannot explicitly ask to view the state information of any particular instance in the network. If this power is added to adversarial capabilities, the models are called 'strong corruption models'. (It is worthwhile to note here that in Reset-1 model, the adversary knows the state information of an instance *only* at the moment of creation of that instance, because he explicitly assigns random coins of his own choice to that instance. On the other hand, in Reset-2 model, the adversary does not know the state information of an instance even at the time of its creation.) More specifically, the following procedure, called **RevealState**, is added to RAKE-1 and RAKE-2.

procedure RevealState(U, i)
If $(U \in \mathcal{U} \vee i \in T_U)$ then return $Invalid$
Timer $\leftarrow$ Timer+1; Time[**RevealState**, (U, i)]$\leftarrow$ Timer
return St_U^i

Due to this additional adversarial power, changes need to be made to the procedure **Finalize** in both the models. This is especially important in the Reset-2 model. Since we allow the adversary A to corrupt the two users U^* and V^* in Reset-2 model, we cannot allow him to see any random coins assigned to $\Pi_{U^*}^{i^*}$ or $\Pi_{V^*}^{j^*}$ during the protocol, otherwise he can compute $ssk_{U^*}^{i^*}$ trivially. The altered procedure incorporates this check wherever the check for the Reveal query occurs. The altered procedure looks as follows.

procedure Finalize(b')
...

If (Time[**Reveal**,(U^*, i^*)] $\vee$ Time[**RevealState**, (U^*, i^*)]) then return false

...

If (Time[**Reveal**,(V^*, j^*)] $\vee$ Time[**RevealState**, (V^*, j^*)]) then return false

...

5 Security Analysis of Existing Protocols

In this section, we show that some of the widely used AKE protocols are insecure under Reset-1 and/or Reset-2 attacks. Section 5.1 gives Reset-1 and Reset-2 attacks on the ISO protocol [1, 5]. Section 5.2 gives Reset-1 and Reset-2 attacks on the SIGMA protocol [6, 13]. Section 5.3 gives a Reset-2 attack on the HMQV [12] protocol that is due to Menezes and Ustaoglu [16].

5.1 Reset-1 and Reset-2 Attacks on the ISO Protocol

The ISO protocol is described in Figure 3. This protocol uses the signature-based Diffie-Hellman paradigm. The signature scheme used is one of those specified in the Digital Signature Standard (FIPS 186-3 [2]). Here, we take our signature scheme to be Digital Signature Algorithm, which is described below.

Digital Signature Algorithm: Let p be a large prime (typically 1024 bits in length), and q a prime divisor of $p-1$ (160 bits in length). Let g be the generator of a subgroup of order q mod p, so $g = h^{\frac{p-1}{q}}$ mod p for some arbitrary $1 < h < p-1$. Let x be the secret signing key known only to the signer such that $0 < x < q$. Let y be the public key, where $y = g^x$ mod p. Let $H(.)$ be a hash function (e.g. SHA-1). Let m be the message to be signed. The signing algorithm consists of the following steps.

- Choose a random number k between 0 and q.
- Compute $r = ((g^k) \bmod p) \bmod q$.
- Compute $s = (k^{-1}(H(m) + xr)) \bmod q$.
- Return the tuple (r, s) as the signature on m.

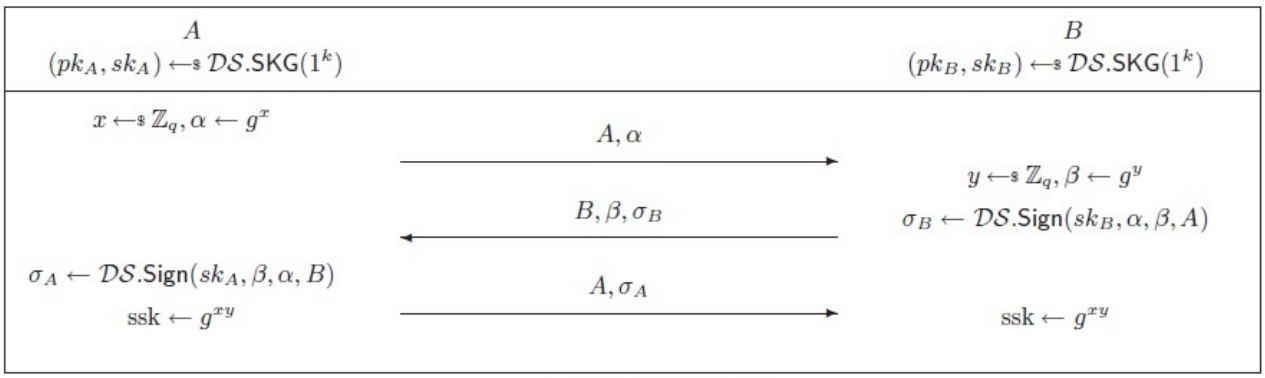


Figure 3: The ISO Protocol (Screenshot from page 9 [18]

Reset-1 Attack: It can be seen that the security of the session key depends *only* on the security of the random numbers x and y because g is public. Since the adversary chooses these values directly for the users, he can trivially compute the session key.

Reset-2 Attack: The goal of the adversary is to make an honest user U sign two different messages (in two different sessions) with the same random parameter. This results in leakage of U's long-lived secret signing key sk_U. Let D be the adversary and let A and B be two honest protocol participants. D carries out the Reset-2 attack as follows.

1. In a session between A and B, D controls the random number k chosen by A while signing the message m_1. So the signature is $(r = g^k, s_1 = k^{-1}(H(m_1) + sk_A r)) \bmod q$.
2. In a second session between A and B, D assigns the same k to A while A is signing the message m_2. The signature is $(r = g^k, s_2 = k^{-1}(H(m_2) + sk_A r)) \bmod q$. Since V is an honest participant, $m_1 \neq m_2$ with high probability. Therefore $s_1 \neq s_2$ with high probability.
3. For D, the equation $\frac{s_1}{s_2} = \frac{H(m_1)+sk_A r}{H(m_2)+sk_A r} \bmod q$ has exactly one unknown sk_A, which can be found by the formula $sk_A = \frac{H(m_1)s_1^{-1} - H(m_2)s_2^{-1}}{rs_2^{-1} - rs_1^{-1}} \bmod q$.

Therefore, the ISO protocol is insecure under both Reset-1 and Reset-2 attacks.

5.2 Reset-1 and Reset-2 Attacks on the SIGMA Protocol

The SIGMA [6, 13] protocol is described in Figure 4. This protocol also uses the signature-based Diffie-Hellman paradigm and we assume our signature scheme to be the Digital Signature Algorithm. Here, x and y are random numbers chosen by the participants A and B, as in the ISO protocol. The session key K_s, the MAC key K_m and the encryption key K_e are all derived from the Diffie-Hellman (DH) value g^{xy}, with the restriction that the three should be pairwise computationally independent (i.e. no information on one can be learned from the other). The purpose of the encryption key is identity protection.

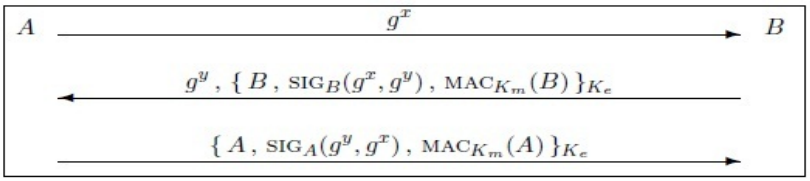


Figure 4: The SIGMA Protocol

Reset-1 Attack: Since the session key K_s is derived from the DH-value g^{xy} whose security depends *only* on the security of x and y, the adversary can trivially compute K_s.

Reset-2 Attack: Since the DH-value is known to the adversary D, D can compute the encryption key K_e to obtain the signatures produced by either of the two parties. Now D makes A sign two different messages using the same random parameter k. As explained in Section 5.1, this results in leakage of the signing key sk_A to D.

Therefore the SIGMA protocol is insecure under both Reset-1 and Reset-2 attacks. In general, any protocol that uses signatures schemes following the Fiat-Shamir paradigm [11] is insecure in the Reset-2 model due to attacks similar to the ones described above. A well-known example is the JFK protocol [3].

5.3 Reset-2 Attack on the HMQV Protocol

The HMQV protocol is described in Figure 5.

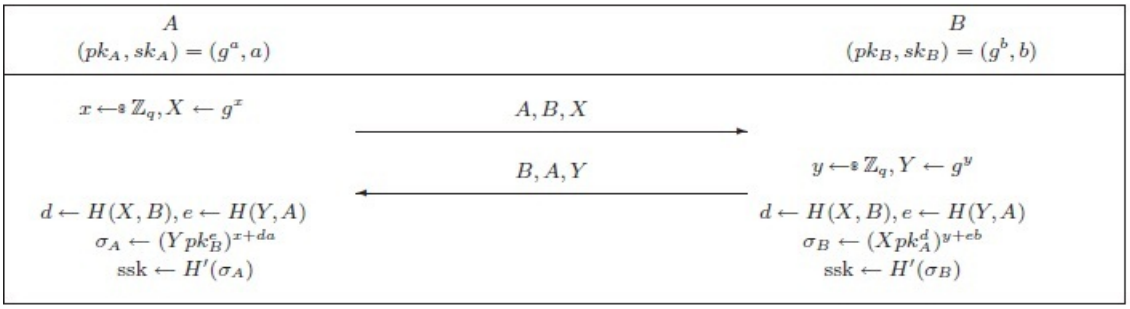


Figure 5: The HMQV Protocol (Screenshot from page 9 [18]

The protocol is implemented in a subgroup G of Z_p^*, where $|G| = q$. A generator of G is g. A and B are ID strings of two users. $H(.)$ and $H'(.)$ are secure hash functions. $x + da$ and $y + eb$ are in Z_q.

We have implemented the original protocol as well as a reset attack on HMQV in Maple. Both $H(.)$ and $H'(.)$ are chosen to be SHA-512 because of its high security. User A and B get random coins from a random source which is a different process. All communications are Maple sockets and the code runs over a local network. More information on how to run the code is in the Readme.txt file of the electronic code submission. The program works for both baby steps and giant steps.

Below are the screenshots of one run of the protocol using baby step.

```
Random Source:
Setting random seed state to 1.

Random Source started from My-PC on port 2526...
Sending random number 1791095846 to Alice...
Sending random number 3093770125 to Bob...
```

Alice:
p=1078819810719733898616179
q=4020067529
g=13734109252404941657517
5 pt User_ID=Alice
Private_Key=1487080512
Public_Key=112922129858467004249048

Alice is listening from My-PC on port 2525...

Initiating a new instance with Bob... Random number : x=1791095846
Sent data : [A=Alice, B=Bob, X=43135584890538308818335]
Received data : [B=Bob, A=Alice, Y=167661104375601076802611]
d=33948
e=64473
sigma_a=382150720436363805945041
Session_Key=5907

Alice is listening from My-PC on port 2525...

Bob:
p=1078819810719733898616179
q=4020067529
g=13734109252404941657517

User_ID=Bob
Private_Key=2381728904
Public_Key=725665444125753304703327

Requesting to start a new instance with Alice...
Received data : [A=Alice, B=Bob, X=43135584890538308818335]
Random number : y=3093770125
Sent data : [B=Bob, A=Alice, Y=167661104375601076802611]
d=33948
e=64473
sigma_a=382150720436363805945041
Session_Key=5907

We now describe a Reset-2 attack on the HMQV protocol. The attack is originally due to Menezes and Ustaoglu [16]. It assumes that $(p-1)/q$ has several small relatively prime factors t_1, t_2, ..., t_n (each less than 40 bits) and that $\prod_{i=1}^{n} t_i > q$. The attack is as follows.

1. The adversary corrupts a user B and obtain B's long-lived secret key b.
2. The adversary activates a new instance of A.

3. After receiving (A, B, X) from A, the adversary choose $Y \in Z_p^*$ of order t_1 and send (B, A, Y) to A.
4. Since the HMQV protocol does not require A to check whether Y belongs to Z_p^*, A computes the session key as $\sigma_A = (Ypk_B^e)^s = Y^s pk_B^{es} = Y^s (g^s)^{be} = Y^s (Xpk_A^d)^{be}$ where $s = x + da \bmod t_1$ and $K = H'(\sigma_A)$.
5. The adversary cannot compute the session key because he does not know the value of y. In the normal protocol, y is chosen randomly and then $Y = g^y$ is computed. In the attack, Y is chosen first; then solving for y is a discrete log problem. However, the adversary can issue a Reveal query against A to get the session key $K = H'(Y^s (Xpk_A^d)^{be})$. Then, the adversary iteratively computes $K' = H'(Y^{c_1} (Xpk_A^d)^{be})$ for $c_1 = 0, 1, 2, ...$ until $K' = K$. Since t_1 has less than 40 bits, it is computational feasible to carry out this brute force attack to find K'. Now $Y \leq t_1$ and $c_1 = s = x + da \bmod t_1$.
6. The adversary now issues a reset-2 attack to reset A. He repeats the attack for $t_2, t_3, ..., t_n$ for the same value of x to compute $c_2, c_3, ..., c_n$.
7. In the HMQV protocol, $s = x + da \bmod q$. The adversary has already computed $c_i = s \bmod t_i$ for $i = 1, 2, ..., n$. The requirement of the attack is $\prod_{i=1}^{n} t_i > q$. Solving these congruences by Chinese Reminder Theorem yields s in Z_q.
8. The adversary now corrupts another party C and repeat steps 1-7 to compute $s' = x + d'a \bmod q$.
9. Obsserve that $d \neq d'$ with high probability because $d = H(X, B)$ and $d' = H(X, C)$, and B and C are different user ID strings. The adversary computes A's private key using the formula $a = (s - s')/(d - d') \bmod q$.

We have implemented this Reset-2 attack in Maple. See the Readme.txt for information on how to run the code. The file 'adversary.mpl' represents the adversary. The adversary is able to send Reset-2 package to the random source to simulate the Reset-2 attack. The attack works for both baby-steps and giant-steps. We give below a few screenshots of the results of the attack that uses giant steps. The size of the parameters are p=1024 bits, q=80 bits and t_i=10 bits.

```
Random Source:
Setting random seed state to 1

Random Source started from My-PC on port 2526...
Simulating RESET2 Attack by resetting the random seed; users will reuse past random numbers.
Sending random number 95775497340659061015434 to Alice...
```

Alice:
p=94871321683986525375611649923190989353700511511688321314708223267910612194069769583359433709792126805718561261287316136273295218795157616679329700532376320755125971374028585153982486541867970528396416388569950719814407585914623820172089354500105245737867615234688879533207774395987689130398182956833921972839
q=1042023093397494815726129
g=2906491663413603730468679889455757048730674283213569317118117354418656236943109592646651361680271325165237003450726945617582110891932061475356247822190842682076668113749404316867583625877119697777999290632221760592148419745562356191198952524857281717916051694695788715597448051725931778350742852586598821645 9

User_ID=Alice
Private_Key=819349744456646322225879
Public_Key=353569997329866323134748875274192652285182803263786204773812331248326512048894568671793625652971984365816016169400388256386714976102748562349214165988270680724512520902794644228966668310802963394344753541430383665245329605314378806390098778558308465679515842746269988992955041609958037420756108641062168726219068189107413770534]
Received data: [B=Bob, A=Alice, Y=259848158271210878284978566348118960224238131985557888133035557642972966691720500348058725798576037552621549359355401663244351653230078515020381029529328481622044392989972507929131129310361276426746618690169751493626960828232682671871065012520519104170361952586070020473118131996850611031989275351401105226077
d=725808913012
e=820891361840
sigma_a=2949028734675208582636988384008657836864723173013427491269445213071410272724865789129398152037932856591839921651790528336026720036166868319996370683146168671431711542119402374530074267792089901720534104712192013271698989359208752422237845669990050902782424592514088904115124518687012947453008212705971046511 8
Session_Key=205457886658

Alice is listening from My-PC on port 2525...

The adversary now computes c_1, ..., c_{10}. We only show screen shots of the first and the last iteration.

Adversary:

Requesting to start a new instance with Alice...

Received data: [A=Alice, B=Bob, X=504144026654575389792482577120606435309048561070648366631029411557709730278547772492792678831949210293782229087710274324221833092891854810657114566746720922281941238090388220619360254424861654791325080305140735939204157492593789560715608129668732053740846087457980829502014312036542918125690618891074137705341

Chosen Y value: Y=259848158271210878284978566348118960224238131985557888133035557642972966691720500348058725798576037552621549359355401663244351653230078515020381029529328481622044392989972507929131129310361276426746618690169751493626960828232682671871065012520519104170361952586070020473118131996850611031989275351401 1 0522607

Sent data: [B=Bob, A=Alice, Y=259848158271210878284978566348118960224238131985557888133035557642972966691720500348058725798576037552621549359355401663244351653230078515020381029529328481622044392989972507929131129310361276426746618690169751493626960828232682671871065012520519104170361952586070020473118131996850611031989275351401 1 0522607

d=725808913012
e=820891361840

Adversary is sending magic power to Alice for revealing the last session key.
Session_Key=205457886658

Start to search c1 in range 1 to 587...
c1=458

.
.
.
.
.

Requesting to start a new instance with Alice...

Received data : [A=Alice, B=Bob, X=5041440266545753897924825771206064353090485610706483666310294115577097302785477724927926788319492102937822290877102743242218330928918548106571145667467209222819412380903882206193602544248616547913250803051407359392041574925937895607156081296687320537408460874579808295020143120365429181256906188910741377053 4]

Chosen Y value : Y=8646173646807833764419190848087670925371406516183209448538038045323528108909912039739854496125456808427808235681370642239709376864987570693611157994263612505496617248921767151209086278446893074128975753763227463485992958123333628765647047007367109737261230051249940120746558785926640133679179926835679141428107

Sent data : [B=Bob, A=Alice, Y=8646173646807833764419190848087670925371406516183209448538080453235281089099120397398544961254568084278082356813706422397093768649875706936111579942636125054966172489217671512090862784468930741289757537632274634859929581233336287656470470073671097372612300512499401207465587859266401336791799268356791416]

d=725808913012

e=517613695431

Adversary is sending magic power to Alice for revealing the last session key.

Session_Key=17888234823

Start to search c10 in range 1 to 613...

c10=487

Using Chinese Remainder Theorem to solve s.

s=808863069151620963121511

d=725808913012

The adversary now corrupts another party Carol and repeats the above attack to compute c1'=376, ..., c10'=93.

Using Chinese Remainder Theorem to solve s'.

s'=47054196940604633906730

d'=1098730398360

Alice private key has been hacked.

Private Key=819349744456646322225879

Running Time Analysis: The machine we used to implement this code is an Intel Core 2 E6300. It takes 0.039802 seconds per iteration to search for c_i in step 5 of the attack. In step 5, there will be $(2^m)/2$ iterations on average, where m is the length of t_i. The running time is bounded by the largest length of t_i. Assume, in the worse case, that the size of each t_i is m bits. Also, assume that q is about

80 bits, as in partice. There will be $\lceil 2 * 80/m \rceil$ iterations for both B and C. The running time is estimated by the formula $f(m) = 0.039802 * \lceil 80/m \rceil * 2^m$, where m is the maximum size of t_i. Below is a table to show the approximate running time of the code in Maple for different values of m.

m	Running Time
10	5 mins
15	2 hours
20	2 days
25	61 days
30	4 years
35	130 years
40	2275 years

The running time is much slower than expected. One reason is that the implementation is done in Maple. If it is done is C, it would be faster. Another reason is that we are using SHA-512 and it is a slow hash function. One simplification we have done is to store $t_1, t_2, ..., t_n$ and the group member with order of t_i in the public key file. In practice, the adversary can obtain this information by brute force because of the small size of t_i. Another simplification we did is to only return the last session key when the adversary is launching the Reveal query. In general, the adversary can reveal any session key. However, it requires us to add a session ID and exchange such information in the protocol. The simplification will work as long as user A is only talking to the adversary during the attack. In summary, the attack requires the adversary to have three special powers. He should be able to

- issue the Reset-2 attack,
- obtain two legitimate users' private keys, and
- reveal each session key from one party.

It is worth mentioning here that for this attack to be successful, the adversary needs a lot of special powers. The attack is unlikely in practice. However, it is not impossible.

6 Transformation from Reset-2 Security to Reset-1 & Reset-2 Security

As we saw in sections 4.2 and 4.3, the Reset-1 and Reset-2 models are incomparable because the former cannot capture the notion of forward secrecy and the latter can. In general, security in the Reset-2 model does not automatically imply security in Reset-1 model. An example is the ISO-R2 protocol shown in Figure 6. Theorem 3 shows that ISO-R2 is Reset-2 secure. However, it is insecure in the Reset-1 model because controlling the random coins directly enables the adversary to compute the session key. In this section, we describe how Reset-2 security can be converted into both Reset-1 and Reset-2 security by doing a simple transformation that involves pseudo-random function families and strong randomness extractors. Therefore, to construct a protocol that is secure in both the models,

one only needs to construct a Reset-2 secure protocol and then apply the transformation to make it Reset-1 secure as well. We first define pseudo-random function families and strong randomness extractors.

Pseudo-Random Function Family: Let $\mathbb{F}$ be a set of efficiently computable functions $f_k : \mathcal{D} \to \mathcal{R}$ from domain $\mathcal{D}$ to range $\mathcal{R}$ such that each function in $\mathbb{F}$ takes a random number $k \in \mathcal{K}$ from the key space as a hidden seed. $\mathbb{F}$ is called a pseudo-random function family if no polynomial time algorithm can distinguish, with high probability, between a truly random function and a function chosen randomly from $\mathbb{F}$. More formally, $\mathbb{F}$ is a pseudo-random function family if for any polynomial time algorithm A, the advantage

$$\mathbf{Adv}^{\text{prf}}_{\mathbb{F},A}(n) = \Pr[A^{f_k(.)}(1^n) = 1] - \Pr[A^{RF(.)}(1^n) = 1] \quad (1)$$

is negligible, where f_k is a function chosen randomly from $\mathbb{F}$, $RF : \mathcal{D} \to \mathcal{R}$ is a truly random function and n is a security parameter.

Strong Randomness Extractors [9]: Strong randomness extraction is a way of converting semi-random input to an output that is close to being truly random. More formally, let $\mathbb{F}$ be a set of efficiently computable functions $f_k : \mathcal{D} \to \mathcal{R}$ from domain $\mathcal{D}$ to range $\mathcal{R}$ such that each function in $\mathbb{F}$ takes a random number $k \in \mathcal{K}$ from the key space as a hidden seed. Let X be a random variable over $\mathcal{D}$ that has min-entropy[4] m (so X is the semi-random input). If k is chosen uniformly at random from $\mathcal{K}$ and r is chosen uniformly at random from $\mathcal{R}$, then $\mathbb{F}$ is called a strong (m, ϵ)-extractor if the statistical distance[5] between the two distributions $k||f_k(X)$[6] and $k||r$ is at most ϵ. For our purpose, we only require $k||f_k(X)$ and $k||r$ to be computationally indistinguishable.

Now we give a transformation for a Reset-2 secure protocol Π. Theorem 1 shows that the resulting protocol Π' is Reset-1 secure. Theorem 2 shows that the incorporation of a strong randomness extractor makes Π' secure in Reset-2 model as well.

The Transformation: Let Π = (SKG, P) be a Reset-2 secure protocol and let $\mathbb{F} = \{f_k : \{0,1\}^{\rho(n)} \to \{0,1\}^{l(n)} | k \in \{0,1\}^{\delta(n)}\}$ be a pseudo-random function family where n is a security parameter, $\rho(n), l(n)$ and $\delta(n)$ are all polynomials of n, and $l(n)$ denotes the maximum number of random bits needed by a user in the execution of P. Construct a new procotol Π'=(SKG', P') as follows.

- SKG': run SKG(1^n) to generate (pk, sk), select $k \leftarrow \{0,1\}^{\delta(n)}$. Set $pk' = pk$ and $sk' = (sk, k)$.
- P': get a $\rho(n)$-bit random string r, then compute $r' \leftarrow f_k(r)$ and run P with the random coins r'.

Essentially, this transformation introduces 'internal randomness' within the protocol. This means that even if the adversary directly or indirectly assigns the random coins of his own choice to a user U, U re-randomizes them by passing them through a pseudo-random function family at the very beginning

[4] Minimum entropy (min-entropy in short) is a measurement of the amount of randomness present in a variable in the worst case. If X is a random variable, the min-entropy of X is the largest number t such that all events occur with probability $\leq 1/2^t$.

[5] Page 12 of http://www.engr.uconn.edu/∼akiayias/cse364fa07/Cryptograph_Primitives_and_Protocols.pdf.

[6] This means the value of k is concatenated with each value of $f_k(X)$.

of the protocol. The resulting random coins are then used throughout the rest of the protocol and they are computationally indistinguishable from truly random coins.

Theorem 1: If Π is a secure AKE protocol in the weak-corruption (strong-corruption) Reset-2 model, and $\mathbb{F}$ is a pseudo-random function family, then Π' is a secure AKE protocol in the weak-corruption (strong corruption) Reset-1 model.

Proof Sketch: We proceed by way of contradiction. Given an adversary A that breaks Π' in the Reset-1 model, we construct another adversary B that breaks Π in the Reset-2 model. Suppose that the session targeted by A is between the users U^* and V^*. Then B's goal is to target the same session in the Reset-2 game. B simulates the Reset-1 game for A by answering any queries related to U^* and V^* using the oracles available to it in the Reset-2 game, and the rest of the queries by doing computations on its own. We show that A does not notice that it is in a simulated game because $\mathbb{F}$ is a pseudo-random function family. Therefore, since A wins the Reset-1 game with high probability by breaking the session between U^* and V^*, B wins the Reset-2 game with high probability by breaking the same session.

Proof: The proof is by contradiction. Let A be an adversary that breaks Π' in the Reset-1 model with non-negligible advantage by targetting a session between users U^* and V^*. We construct another adversary B who breaks Π in the Reset-2 model with non-negligible advantage. We do this by first constructing a restricted adversary A_1 against Π' in RAKE-1 who can only target sessions between U^* and V^* (and hence it cannot use the Corrupt query on U^* and V^*). We then modify RAKE-1 to a new game G^1, and calculate A_1's advantage against Π' in G^1. We then modify G^1 to obtain a new game G^2 and compute A_1's advantage against Π' in G^2. Finally we let B simulate the game G^2 for A_1, and show that B can win RAKE-2 with non-negligible advantage.

Construct a restricted adversary A_1, from A, against Π' as follows. After the initialization phase, A_1 outputs two distinct user IDs U^* and V^* from the set $\mathcal{U}$. A_1 now simulates RAKE-1 for A as follows. A_1 answers all the queries made by A using its own oracles, and if A invokes the Corrupt query on U^* or V^*, then A_1 inputs a random bit to the Finalize procedure and aborts. Let E denote the event that A invokes the Test query with input (I^*, j^*) such that $(I^*, pid_{I^*}^{j^*}) = (U^*, V^*)$. If E does not occur, then A_1 inputs a random bit to the Finalize procedure and aborts. If E occurs, then A_1 invokes its own Test oracle-query with the same input and returns the response it gets to A. Finally, when A inputs a bit b' to the Finalize procedure, A_1 also inputs b' to the Finalize procedure and aborts.

Let m be the total number of users in the network. Observe that if E does not occur, the probabilty of A_1 winning RAKE-1 against Π' is 0.5. Moreover, U^*, V^* are chosen randomly by A_1, so $\Pr[E]=\frac{1}{m(m-1)}$. Therefore

$$\begin{aligned}
\mathbf{Adv}^{\text{rake-1}}_{\Pi',A_1}(n) &= \Pr[\text{RAKE-1}_{\Pi',A_1}(n) \to \text{true}|E]\Pr[E] + \Pr[\text{RAKE-1}_{\Pi',A_1}(n) \to \text{true}|\neg E]\Pr[\neg E] - \frac{1}{2} \\
&= \Pr[\text{RAKE-1}_{\Pi',A_1}(n) \to \text{true}|E]\Pr[E] + \frac{1}{2}\Pr[\neg E] - \frac{1}{2} \\
&= \Pr[\text{RAKE-1}_{\Pi',A_1}(n) \to \text{true}|E]\Pr[E] + \frac{1}{2}(1 - \Pr[E]) - \frac{1}{2} \\
&= (\Pr[\text{RAKE-1}_{\Pi',A_1}(n) \to \text{true}|E] - \frac{1}{2})\Pr[E] \\
&\geq (\Pr[\text{RAKE-1}_{\Pi',A}(n) \to \text{true}] - \frac{1}{2})\Pr[E] \because \text{if } E \text{ occurs and } A \text{ wins, then } A_1 \text{ also wins} \\
&= \frac{1}{m(m-1)}\mathbf{Adv}^{\text{rake-1}}_{\Pi',A}(n) \qquad (2)
\end{aligned}$$

Now we modify RAKE-1 for Π' by replacing the function $f_{k^*_U}(.)$ with a truly random function RF(.), where $f_{k_{U^*}}(.)$ is the pseudo-random function (belonging to the pseudo-random function family $\mathbb{F}$) with key k_{U^*} used by U^* in the RAKE-1. Call this game G^1.

Claim 1.1: The difference between A_1's advantages in RAKE-1 and G^1 is negligible.

Proof: We show that if this difference is non-negligible, then we can construct an adversary D such that $\mathbf{Adv}^{\text{prf}}_{\mathbb{F},D}(n)$ is non-negligible, therefore it can break the pseudo-random function family.

Let D be an adversary against $\mathbb{F}$ with access to an oracle $\mathcal{O}$, which is either a truly random function RF(.) or a pseudo-random function $f_k(.)$. D wishes to determine which of these is $\mathcal{O}$. To achieve this goal, D simulates RAKE-1 for A_1 by using A_1's oracles to answer all of A_1's queries, except that D simulates the pseudo-random function $f_{k_{U^*}}(.)$ of U^* by asking its own oracle $\mathcal{O}$. If A_1 wins RAKE-1, D outputs 1 and aborts, otherwise D aborts without any output. By (1), we have

$$\begin{aligned}
\mathbf{Adv}^{\text{prf}}_{\mathbb{F},D}(n) &= \Pr[D^{f_k(.)}(1^n) = 1] - \Pr[D^{\text{RF}(.)}(1^n) = 1] \\
&= \Pr[A_1\text{wins the game}|\mathcal{O} = f_k(.)] - \Pr[A_1\text{wins the game}|\mathcal{O} = \text{RF}] \\
&= \Pr[\text{RAKE-1}_{\Pi',A_1}(n) \to \text{true}] - \Pr[G^1_{\Pi',A_1}(n) \to \text{true}] \\
&= \mathbf{Adv}^{\text{rake-1}}_{\Pi',A_1}(n) - \mathbf{Adv}^{G^1}_{\Pi',A_1}(n).
\end{aligned}$$

We see from this equation that if the difference between A_1's advantages in the two games is non-negligible, then D has a non-negligible advantange in breaking the pseudo-random function family, a contradiction. This completes the proof of the claim. □

Now we modify G^1 for Π' by replacing the function $f_{k^*_V}(.)$ with a truly random function RF(.), where $f_{k_{V^*}}(.)$ is the pseudo-random function (belonging to $\mathbb{F}$) with key k_{V^*} used by V^* in G^1. Call this game G^2.

Claim 1.2: The difference between A_1's advantages in G^1 and G^2 is negligible.

Proof: We show that if this difference is non-negligible, then we can construct an adversary D such that $\mathbf{Adv}^{\text{prf}}_{\mathbb{F},D}(n)$ is non-negligible, therefore it can break the pseudo-random function family.

Let D be an adversary against $\mathbb{F}$ with access to an oracle $\mathcal{O}$, which is either a truly random function RF(.) or a pseudo-random function $f_k(.)$. D wishes to determine which of these is $\mathcal{O}$. To achieve

this goal, D simulates G^1 for A_1 by using A_1's oracles to answer all of A_1's queries, except that D simulates the pseudo-random function $f_{k_{V^*}}(.)$ of V^* by asking its own oracle $\mathcal{O}$. If A_1 wins G^1, D outputs 1 and aborts, otherwise D aborts without any output. By (1), we have

$$\begin{aligned}\mathbf{Adv}^{\mathrm{prf}}_{\mathbb{F},D}(n) &= \Pr[D^{f_k(.)}(1^n) = 1] - \Pr[D^{\mathrm{RF}(.)}(1^n) = 1] \\ &= \Pr[A_1\text{wins the game}|\mathcal{O} = f_k(.)] - \Pr[A_1\text{wins the game}|\mathcal{O} = \mathrm{RF}] \\ &= \Pr[G^1_{\Pi',A_1}(n) \to \text{true}] - \Pr[G^2_{\Pi',A_1}(n) \to \text{true}] \\ &= \mathbf{Adv}^{G^1}_{\Pi',A_1}(n) - \mathbf{Adv}^{G^2}_{\Pi',A_1}(n).\end{aligned}$$

We see from this equation that if the difference between A_1's advantages in the two games is non-negligible, then D has a non-negligible advantange in breaking the pseudo-random function family, a contradiction. This completes the proof of the claim. □

Now given an adversary A_1 against Π' in game G^2, we construct an adversary B against Π in the game RAKE-2 that simulates G^2 for A_1. B's aim is to target the session between U^* and V^*. In the Initialization phase for A_1, B gives the public keys of all the users to A_1. After A_1 outputs U^* and V^*, B uses the Corrupt query to learn the long-lived secret keys of all the users in the set $\mathcal{U}\backslash\{U^*, V^*\}$ (note that A_1 never invokes the Corrupt query on U^* and V^*, so B does not need to corrupt these two). B also chooses $k_J \leftarrow \mathcal{K}$ for all $J \in \mathcal{U}\backslash\{U^*, V^*\}$; these are the keys that are part of the secret keys used in Π'. Now B simulates the game G^2 for A_1 as follows.

- When A_1 makes a Register query with input (I, pk_I), B makes a Register query to its own oracle with the same input.
- When A_1 makes a NewInstance query with input (I, j, N) for some user I,
  (a) if $I \notin \{U^*, V^*\}$, B performs the operations in the NewInstance procedure of G^2 itself.
  (b) if $I \in \{U^*, V^*\}$, B checks if A_1 has ever made a NewInstance query with input (I, j', N) where $j' \neq j$ (basically, B is checking if it needs to reset an instance of I). If it has, B makes a NewInstance query with input $(I, j, N_{j'})$ where $N_{j'}$ are the random coins assigned to $\Pi^{j'}_I$. If not, B makes a NewInstance query with input $(I, j, \perp)$, where $\perp$ denotes any randomly chosen coins.
- When A_1 makes a Send query with input (I, j, M_{in}),
  (a) if $I \notin \{U^*, V^*\}$, B performs the operations in the Send procedure of G^2 itself (B invokes the protocol P' using k_I as the key of the pseudo-random function).
  (b) if $I \in \{U^*, V^*\}$, B makes a Send query to its own oracle with the same input and returns to A_1 whatever response it gets.
- When A_1 makes a Reveal or RevealState query with input (I, j), B answers them in a way exactly analogous to the Send query.
- When A_1 makes a Corrupt query, it cannot be on U^* or V^*. Since B has corrupted every other user, it answers this query using its own computations.
- When A_1 makes a Test query with input (I, j), then $(I, pid^j_I) = (U^*, V^*)$. Then B makes a Test query to its own oracle with the same input and returns to A_1 whatever response it gets.

- When A_1 inputs a bit b' to the Finalize procedure, B also inputs b' to the Finalize procedure in RAKE-2 and aborts.

The way B simulates G^2 for A_1, there is no way for A_1 to know that it is simulated. Also, A_1 and B are attacking the same session, so if A_1's input to the Test query clears all the if-conditions in the Finalize procedure of G_2, then B's input to the Test query also clears all the if-conditions in the Finalize procedure of RAKE-2. So the probability of B winning RAKE-2 is at least the probability of A_1 winning G^2. Hence

$$\begin{aligned}
\mathbf{Adv}^{\text{rake-2}}_{\Pi,B}(n) &= \Pr[\text{RAKE-2}_{\Pi,B}(n) \to \text{true}] - \frac{1}{2} \\
&\geq \Pr[G^2_{\Pi',A_1}(n) \to \text{true}] - \frac{1}{2} \\
&= \mathbf{Adv}^{G^2}_{\Pi',A_1}(n) \\
&= \mathbf{Adv}^{G^1}_{\Pi',A_1}(n) - \mathbf{Adv}^{\text{prf}}_{\mathbb{F},D}(n) \text{ by Claim 1.2} \\
&= \mathbf{Adv}^{\text{rake-1}}_{\Pi',A_1}(n) - 2\mathbf{Adv}^{\text{prf}}_{\mathbb{F},D}(n) \text{ by Claim 1.1} \\
&\geq \frac{1}{m(m-1)}\mathbf{Adv}^{\text{rake-1}}_{\Pi',A}(n) - 2\mathbf{Adv}^{\text{prf}}_{\mathbb{F},D}(n) \text{ by (2)}
\end{aligned}$$

By assumption, $\mathbf{Adv}^{\text{rake-1}}_{\Pi',A}(n)$ is non-negligible, so there exists a positive integer c such that $|\mathbf{Adv}^{\text{rake-1}}_{\Pi',A}(n)| \geq \frac{1}{n^c}$ for sufficiently large n. Since $\mathbf{Adv}^{\text{prf}}_{\mathbb{F},D}(n)$ is negligible, we have $|\mathbf{Adv}^{\text{rake-2}}_{\Pi,B}(n)| \geq \frac{1}{m(m-1)}|\mathbf{Adv}^{\text{rake-1}}_{\Pi',A}(n)| \geq \frac{1}{m(m-1)}\frac{1}{n^c} \geq \frac{1}{n^a}$ for a sufficiently large a. So $\mathbf{Adv}^{\text{rake-2}}_{\Pi,B}(n)$ is non-negligible, a contradiction. This completes the proof of Theorem 1. □

Theorem 1 says that given a Reset-2 secure protocol, we can obtain a Reset-1 secure protocol by applying the transformation. In general, however, the new protocol may not be secure in the Reset-2 model. More specifically, in the Reset-2 model, the adversary is allowed to corrupt the long-lived secret key $sk'_{U^*} = (sk_{U^*}, k_{U^*})$ of the user U^* that he inputs to the Test query. Since the security of the pseudo-random function $f_{k_{U^*}}$ relies on the secrecy of k_{U^*} and k_{U^*} is known to the adversary, we cannot assume that the output of $f_{k_{U^*}}$ is still computationally indistinguishable from truly random coins. So even if the input r to $f_{k_{U^*}}$ is truly random, the output $f_{k_{U^*}}(r)$ may not be random from the point of view of the adversary. To solve this problem, we add the condition that the $\mathbb{F}$ used in the transformation should be a strong randomness extractor. So now, even if the adversary knows k_{U^*}, the distribution $k_{U^*}||f_{k_{U^*}}$ is computationally indistinguishably from a truly random distribution. The next theorem shows that the transformed protocol is secure in the Reset-2 model, provided that the pseudo-random function family is also a strong randomness extractor.

Theorem 2: If Π is a secure AKE protocol in the weak-corruption (strong-corruption) Reset-2 model and $\mathbb{F}$ is a pseudo-random function family and a strong randomness extractor, then Π' is secure in the weak-corruption (strong-corruption) Reset-2 model.

Proof Sketch: We proceed by way of contradiction. Given an adversary A that breaks Π' in the Reset-2 model, we construct another adversary B that breaks Π in the Reset-2 model. Suppose that the session key targeted by A is between the users U^* and V^*. Then B's goal is to target the

same session key in the Reset-2 game. We break the proof into two cases, depending on whether the instance input by A into the Test query has a partner instance or not. In each case, we first construct a restricted adversary A_1 against Π' in RAKE-2 who can only target sessions between U^* and V^*. We then modify RAKE-2 to a new game G^1, and calculate A_1's advantage against Π' in G^1. We then modify G^1 to obtain a new game G^2 and compute A_1's advantage against Π' in G^2. Finally we let B simulate the game G^2 for A_1. We show that A_1 does not notice that it is in a simulated game because $\mathbb{F}$ is a pseudo-random function family and a strong randomness extractor. Therefore, if A wins RAKE-1 against Π' with high probability by breaking the session between U^* and V^*, B wins RAKE-2 against Π with high probability by breaking the same session.

Proof: By way of contradiction, let A be an adversary that can break Π' in the Reset-2 model. We distinguish two cases.

Case 1: the instance input by A into the Test query has a partner instance. In this case, we follow the same method as in Theorem 1's proof. We construct a restricted adversary A_1, from A, against Π' as follows. After the initialization phase, A_1 outputs two distinct numbers l and l' such that the target session is between the l-th and l'-th instances (let these instances be (U^*, i^*) and (V^*, j^*) respectively). A_1 now simulates RAKE-2 for A as follows. A_1 answers all the queries made by A using its own oracles. Let E denote the event that A invokes the Test query with input (I^*, c^*) such that $(I^*, c^*) = (U^*, i^*)$ or (V^*, j^*) . If E does not occur, then A_1 inputs a random bit to the Finalize procedure and aborts. If E occurs, then A_1 invokes the its own Test oracle-query with the same input and returns the response it gets to A. Finally, when A inputs a bit b' to the Finalize procedure, A_1 also inputs b' to the Finalize procedure and aborts.

Let q be the total number of NewInstance queries made by A. Observe that if E does not occur, the probabilty of A_1 winning RAKE-2 against Π' is 0.5. Moreover, the instances l, l' are chosen randomly by A_1, so $\Pr[E]=\frac{1}{q(q-1)}$. Therefore

$$\begin{aligned}
\mathbf{Adv}^{\text{rake-2}}_{\Pi',A_1}(n) &= \Pr[\text{RAKE-2}_{\Pi',A_1}(n) \rightarrow \text{true}|E]\Pr[E] + \Pr[\text{RAKE-2}_{\Pi',A_1}(n) \rightarrow \text{true}|\neg E]\Pr[\neg E] - \frac{1}{2} \\
&= \Pr[\text{RAKE-2}_{\Pi',A_1}(n) \rightarrow \text{true}|E]\Pr[E] + \frac{1}{2}(1 - \Pr[E]) - \frac{1}{2} \\
&= (\Pr[\text{RAKE-2}_{\Pi',A_1}(n) \rightarrow \text{true}|E] - \frac{1}{2})\Pr[E] \\
&\geq (\Pr[\text{RAKE-2}_{\Pi',A}(n) \rightarrow \text{true}] - \frac{1}{2})\Pr[E] \because \text{if } E \text{ occurs and } A \text{ wins, then } A_1 \text{ also wins} \\
&= \frac{1}{q(q-1)}\mathbf{Adv}^{\text{rake-2}}_{\Pi',A}(n) \qquad (3)
\end{aligned}$$

Now we modify RAKE-2 for Π' to a new game G^1 such that in the instance (U^*, i^*), the output of the function $f_{k_{U^*}}(.)$ is replaced with a truly random function RF(.), where k_{U^*} is the key used by U^* in RAKE-2 and $f_{k_{U^*}}(.) \in \mathbb{F}$. Call this game G^1.

Claim 2.1: The difference between A_1's advantages in RAKE-2 and G^1 is negligible.

Proof: We show that if this difference is non-negligible, then we can construct an adversary D such that $\mathbf{Adv}^{\text{prf}}_{\mathbb{F},D}(n)$ is non-negligible, therefore it can break the pseudo-random function family and strong

randomness extractor.

Let D be an adversary against $\mathbb{F}$ with access to an oracle $\mathcal{O}$, which is either a truly random function RF(.) or $f_k(.) \in \mathbb{F}$. D wishes to determine which of these is $\mathcal{O}$. To achieve this goal, D simulates RAKE-2 for A_1 by using A_1's oracles to answer all of A_1's queries, except that D simulates the function $f_{k_{U^*}}(.)$ of U^* by asking its own oracle $\mathcal{O}$. If A_1 wins RAKE-2, D outputs 1 and aborts, otherwise D aborts without any output. Since $\mathbb{F}$ is a pseudo-random function family as well as a (m, ϵ)-strong randomness extractor, we have $\mathbf{Adv}^{\text{sre}}_{\mathbb{F},D}(n) \leq \epsilon$. By (1),

$$\begin{aligned}
\mathbf{Adv}^{\text{sre}}_{\mathbb{F},D}(n) &= \Pr[D^{f_k(.)}(1^n) = 1] - \Pr[D^{\text{RF}(.)}(1^n) = 1] \\
\epsilon &\geq \Pr[A_1\text{wins the game}|\mathcal{O} = f_k(.)] - \Pr[A_1\text{wins the game}|\mathcal{O} = \text{RF}] \\
&= \Pr[\text{RAKE-2}_{\Pi',A_1}(n) \to \text{true}] - \Pr[G^1_{\Pi',A_1}(n) \to \text{true}] \\
&= \mathbf{Adv}^{\text{rake-2}}_{\Pi',A_1}(n) - \mathbf{Adv}^{G^1}_{\Pi',A_1}(n).
\end{aligned}$$

So if the difference between A_1's advantages in the two games is non-negligible, then D has a non-negligible advantage in breaking $\mathbb{F}$, a contradiction. □

Now we modify G^1 for to a new game G^2 such that in the instance (V^*, j^*), the output of the function $f_{k_{V^*}}(.)$ is replaced with a truly random function RF(.), where k_{V^*} is the key used by V^* in G^1 and $f_{k_{V^*}}(.) \in \mathbb{F}$. Call this game G^2.

Claim 2.2: The difference between A_1's advantages in G^1 and G^2 is negligible.

Proof: We show that if this difference is non-negligible, then we can construct an adversary D such that $\mathbf{Adv}^{\text{sre}}_{\mathbb{F},D}(n)$ is non-negligible, therefore it can break $\mathbb{F}$.

Let D be an adversary against $\mathbb{F}$ with access to an oracle $\mathcal{O}$, which is either a truly random function RF(.) or $f_k(.) \in \mathbb{F}$. D wishes to determine which of these is $\mathcal{O}$. To achieve this goal, D simulates G^1 for A_1 by using A_1's oracles to answer all of A_1's queries, except that D simulates the function $f_{k_{V^*}}(.)$ of V^* by asking its own oracle $\mathcal{O}$. If A_1 wins G^1, D outputs 1 and aborts, otherwise D aborts without any output. By (1), we have

$$\begin{aligned}
\mathbf{Adv}^{\text{sre}}_{\mathbb{F},D}(n) &= \Pr[D^{f_k(.)}(1^n) = 1] - \Pr[D^{\text{RF}(.)}(1^n) = 1] \\
\epsilon &\geq \Pr[A_1\text{wins the game}|\mathcal{O} = f_k(.)] - \Pr[A_1\text{wins the game}|\mathcal{O} = \text{RF}] \\
&= \Pr[G^1_{\Pi',A_1}(n) \to \text{true}] - \Pr[G^2_{\Pi',A_1}(n) \to \text{true}] \\
&= \mathbf{Adv}^{G^1}_{\Pi',A_1}(n) - \mathbf{Adv}^{G^2}_{\Pi',A_1}(n).
\end{aligned}$$

So if the difference between A_1's advantages in the two games is non-negligible, then D has a non-negligible advantange in breaking the pseudo-random function family, a contradiction. □

Now given an adversary A_1 against Π' in G^2, construct an adversary B against Π in RAKE-2. B corrupts all the parties in $\mathcal{U}$, selects $k_J \in \mathcal{K}$ for all $J \in \mathcal{U}$ and simulates the game G^2 for A_1 as follows. For all the queries made by A_1 related to instances other than the l-th and l'-th instances, B computes the response itself. For queries relates to l-th and l'-th instances, B relays these queries to its own oracle and returns to A_1 the responses it receives. Finally when A_1 inputs a bit b' to the Finalize procedure, B also inputs b' to its own Finalize procedure and aborts.

The way B simulates G^2 for A_1, there is no way for A_1 to know that it is simulated. Also, A_1 and B are attacking the same session, so if A_1's input to the Test query clears all the if-conditions in the Finalize procedure of G_2, then B's input to the Test query also clears all the if-conditions in the Finalize procedure of RAKE-2. So the probability of B winning RAKE-2 is at least the probability of A_1 winning G^2. Hence

$$\begin{aligned}
\mathbf{Adv}^{\text{rake-2}}_{\Pi,B}(n) &= \Pr[\text{RAKE-2}_{\Pi,B}(n) \to \text{true}] - \frac{1}{2} \\
&\geq \Pr[G^2_{\Pi',A_1}(n) \to \text{true}] - \frac{1}{2} \\
&= \mathbf{Adv}^{G^2}_{\Pi',A_1}(n) \\
&\geq \mathbf{Adv}^{G^1}_{\Pi',A_1}(n) - \epsilon \ \text{ by Claim 2.2} \\
&\geq \mathbf{Adv}^{\text{rake-2}}_{\Pi',A_1}(n) - 2\epsilon \ \text{ by Claim 2.1} \\
&\geq \frac{1}{q(q-1)}\mathbf{Adv}^{\text{rake-2}}_{\Pi',A}(n) - 2\epsilon \ \text{ by (3)}
\end{aligned}$$

By assumption, $\mathbf{Adv}^{\text{rake-2}}_{\Pi',A}(n)$ is non-negligible, so there exists a positive integer c such that $|\mathbf{Adv}^{\text{rake-2}}_{\Pi',A}(n)| \geq \frac{1}{n^c}$ for sufficiently large n. Since ϵ is negligible, we have $|\mathbf{Adv}^{\text{rake-2}}_{\Pi,B}(n)| \geq \frac{1}{q(q-1)}|\mathbf{Adv}^{\text{rake-2}}_{\Pi',A}(n)| \geq \frac{1}{q(q-1)}\frac{1}{n^c} \geq \frac{1}{n^a}$ for a sufficiently large a. So $\mathbf{Adv}^{\text{rake-2}}_{\Pi,B}(n)$ is non-negligible, a contradiction. This completes Case 1.

Case 2: the instance input by A into the Test query has no partner instance. In this case, similar to Case 1, we first define a restricted adversary A_1 that outputs an integer l and an identity V^* after the Initialize phase such that A_1 inputs the l-th instance (denoted by (U^*, i^*)) in the Test query and $pid^{i^*}_{U^*} = V^*$. A_1 now simulates RAKE-2 for A as follows. A_1 answers all the queries made by A using its own oracles. Let E denote the event that A invokes the Test query with input (I^*, c^*) such that $(I^*, pid^{c^*}_{I^*}) = (U^*, V^*)$. If E does not occur, then A_1 inputs a random bit to the Finalize procedure and aborts. If E occurs, then A_1 invokes the its own Test oracle-query with the same input and returns the response it gets to A. Finally, when A inputs a bit b' to the Finalize procedure, A_1 also inputs b' to the Finalize procedure and aborts.

Let q be the total number of NewInstance queries made by A and let $m = |\mathcal{U}|$. Since the instance (U^*, i^*) and ID V^* are chosen randomly by A_1, $\Pr[E]=\frac{1}{mq}$. Then by analysis exactly the same as that in Case 1, we have

$$\mathbf{Adv}^{\text{rake-2}}_{\Pi',A_1}(n) \geq \frac{1}{mq}\mathbf{Adv}^{\text{rake-2}}_{\Pi',A}(n). \tag{4}$$

We modify RAKE-2 to a new game G^1 such that in the instance (U^*, i^*), we replace the function $f_{k_{U^*}}(.)$ by a truly random function. Then by the same analysis as that in Case 1, we have

$$\mathbf{Adv}^{G^1}_{\Pi',A_1}(n) \geq \mathbf{Adv}^{\text{rake-2}}_{\Pi',A_1}(n) - \epsilon. \tag{5}$$

Next, we modify G^1 to a new game G^2 such that we replace the function $f_{k_{U^*}}(.)$ by a truly random function. Then by the same analysis as that in Theorem 1, we have

$$\mathbf{Adv}^{G^2}_{\Pi',A_1}(n) \geq \mathbf{Adv}^{G^1}_{\Pi',A_1}(n) - \mathbf{Adv}^{\text{prf}}_{\mathbb{F},D}(n). \tag{6}$$

Finally, given an adversary A_1 against protocol Π' in game G^2, we construct another adversary B against Π in RAKE-2 as follows. B corrupts all the users in the set $\mathcal{U}\backslash\{V^*\}$ and learns their corresponding long-lived private keys. B also chooses $k_J \in \mathcal{K}$ for all $J \in \mathcal{U}\backslash\{V^*\}$. B answers itself all the queries made by A except the following:

1. For all the queries related to the instance (U^*, i^*) including the Test query, B relays the queries to its own oracle and returns to A_1 the responses it receives.
2. For all the queries related to user V^*, B relays the queries to its own oracle and returns to A_1 the responses it receives. Note that A_1 never makes a Corrupt query on V^*.

At the end, when A_1 inputs a bit b' to the Finalize procedure and aborts, B also inputs b' to its Finalize procedure and aborts. Then, we have

$$\begin{aligned}
\mathbf{Adv}^{\text{rake-2}}_{\Pi,B}(n) &= \Pr[\text{RAKE-2}_{\Pi,B}(n) \to \text{true}] - \frac{1}{2} \\
&\geq \Pr[G^2_{\Pi',A_1}(n) \to \text{true}] - \frac{1}{2} \\
&= \mathbf{Adv}^{G^2}_{\Pi',A_1}(n) \\
&\geq \mathbf{Adv}^{G^1}_{\Pi',A_1}(n) - \mathbf{Adv}^{\text{prf}}_{\mathbb{F},D}(n) \text{ by (6)} \\
&\geq \mathbf{Adv}^{\text{rake-2}}_{\Pi',A_1}(n) - \epsilon - \mathbf{Adv}^{\text{prf}}_{\mathbb{F},D}(n) \text{ by (5)} \\
&\geq \frac{1}{mq}\mathbf{Adv}^{\text{rake-2}}_{\Pi',A}(n) - \epsilon - \mathbf{Adv}^{\text{prf}}_{\mathbb{F},D}(n) \text{ by (4)}
\end{aligned}$$

By assumption, $\mathbf{Adv}^{\text{rake-2}}_{\Pi',A}(n)$ is non-negligible, so by non-negligibility analysis similar to that in Case 1 and in Theorem 1, $\mathbf{Adv}^{\text{rake-2}}_{\Pi,B}(n)$ is non-negligible, a contradiction. This completes Case 2 and the proof of Theorem 2. □

7 Two New Reset-1 and Reset-2 Secure Protocols

In this section, we modify the ISO procotol and the SKEME protocol [14] to obtain new protocols that are Reset-1 and Reset-2 secure. Section 7.1 describes a new protocol based on ISO, gives the proof of its Reset-2 security and then gives a transformed protocol that is Reset-1 secure as well. Section 7.2 describes a new protocol based on SKEME, gives the proof of its Reset-2 security and then gives a transformed Reset-1 secure protocol. We begin with some definitions.

A digital signature scheme $\mathcal{DS}$, which consists of a signing algorithm $\mathcal{DS}$.Sign and a verification algorithm $\mathcal{DS}$.Verify, is a **Deterministic Digital Signature Scheme** if $\mathcal{DS}$.Sign is deterministic. Any randomized digital signature scheme can be converted to a deterministic one as follows: the signing key is expanded to include a key k' which is chosen uniformly at random from the key space of the pseudo-random function family $\mathbb{F}'$. To sign a message m, we first compute random coins $r = f'_{k'}(m)$, where $f'_{k'} \in \mathbb{F}'$, and then invoke the (randomized) signing algorithm $\mathcal{DS}$.Sign with random coins r.

A digital signature scheme $\mathcal{DS}$ is **Existentially Unforgeable under Adaptive Chosen Message Attacks** (i.e. it is uf-cma) if any adversary who is given q valid (message, signature) pairs cannot

generate a $(q+1)$'th valid pair without knowing the secret signing key or without using the signing algorithm. More formally, $\mathcal{DS}$ is uf-cma, if for any polynomial time algorithm F,

$$\mathbf{Adv}^{\text{uf-cma}}_{\mathcal{DS},F}(n) = \Pr\Big[(pk, sk) = \mathcal{DS}.\text{SKG}(1^n), (m^*, \sigma^*) \to F^{\mathcal{DS}.\text{Sign}(sk,.)}(pk) : \\ \mathcal{DS}.\text{Verify}(pk, m^*, \sigma^*) = 1 \wedge F \text{ has never queried } \mathcal{DS}.\text{Sign}(sk, m^*)\Big]$$

is negligible.

The **Decisional Diffie-Hellman (DDH) Assumption** says that for any polynomial time algorithm F,

$$\mathbf{Adv}^{\text{DDH}}_{F}(n) = \Pr[F(1^n, g, g^a, g^b, Z) = 1 | Z = g^{ab}] - \Pr[F(1^n, g, g^a, g^b, Z) = 1 | Z = g^r]$$

is negligible, where a, b, r are randomly selected from $\mathbb{Z}_p$ and p is a prime. Simply put, the DDH assumption means that given the values $g, g^a, g^b \in \mathbb{Z}_p$, an adversary cannot determine with high probability whether some number $Z \in \mathbb{Z}_p$ equals g^{ab} or not.

A **Public Key Encryption Scheme** $\mathcal{PKE}$ consists of three algorithms. The key generation algorithm $\mathcal{PKE}.\text{SKG}(1^n)$ takes a security parameter n as input and outputs a public/secret key pair (pk, sk). The encryption algorithm $\mathcal{PKE}.\text{Enc}(pk, m)$ takes the public key and a message as input and outputs a ciphertext c. The decryption algorithm $\mathcal{PKE}.\text{Dec}(sk, c)$ takes the secret key and a ciphetext as input and outputs m or $\perp$ (which indicates decryption failure). A public key encryption scheme $\mathcal{PKE}$ is secure under adaptive Chosen Ciphertext Attacks (CCA) if for any PPT adversary $A = (A_1, A_2)$,

$$\mathbf{Adv}^{\text{cca}}_{\mathcal{PKE},A}(n) = \Pr\Big[(pk, sk) \leftarrow \mathcal{PKE}.\text{SKG}(1^n), (x_0, x_1, \delta) \leftarrow A_1^{\mathcal{PKE}.\text{Dec}(sk,.)}(pk), b \xleftarrow{\$} \{0,1\}, \\ y \leftarrow \mathcal{PKE}.\text{Enc}(pk, x_b), b' \leftarrow A_2^{\mathcal{PKE}.\text{Dec}(sk,.)}(pk, x_0, x_1, \delta, y) : b' = b\Big] - \frac{1}{2}$$

is negligible, where $|x_0| = |x_1|$, and A_2 is not allowed to make a decryption query with input y. Simply put, the adversary A is given access to an encryption oracle and a decryption oracle and he may perform any number of encryption or decryption operations on arbitrary plaintexts and ciphertexts. The adversary chooses two distinct plaintexts x_0 and x_1 and sends them to a challenger, who selects a bit b randomly, encrypts x_b and sends the result y back to A. A then performs as many decryptions and encryptions as it wants, except using the decrypt query on y. It then outputs a bit b', which is his guess for the value of b. The scheme is secure under adaptive CCA if A cannot correctly guess the value of b with high probability.

A **Message Authentication Code** scheme $\mathcal{MAC}$ with key space $\mathcal{K}$ consists of two algorithms. $\text{MAC}(K, m)$ is a message authentication algorthm that takes a key $K \in \mathcal{K}$ and a message m as input and returns an authentication tag τ. $\text{MAV}(K, m, \tau)$ is a verification algorithm that takes a key $K \in \mathcal{K}$, a message m, and an authentication tag τ as input. It returns 1 if $\tau = \text{MAC}(K, m)$ and 0 otherwise. A message authentication code scheme $\mathcal{MAC}$ is secure under adaptive Chosen Message Attacks (CMA), if for any polynomial time algorithm A,

$$\mathbf{Adv}^{\text{cma}}_{\mathcal{MAC},A}(n) = \Pr\Big[K \xleftarrow{\$} \mathcal{K}, (m^*, \tau^*) \leftarrow A^{\text{MAC}(K,.)}(1^n) : \\ \text{MAV}(K, m^*, \tau^*) = 1 \wedge A \text{ has never queried MAC}(K, m^*)\Big]$$

is negligible. Simply put, given a key $K \in \mathcal{K}$, the goal of the adversary A is to produce an authentication tag τ^* on a message m^* without quering MAC, such that MAV accepts this tag and outputs 1. The scheme is secure under adaptive CMA if A cannot succeed with high probability.

7.1 An ISO-based Reset-1 and Reset-2 Secure Protocol

We saw in Section 5.1 that the ISO protocol (Figure 1) is insecure under both the attacks. Yang et al. [18] made two modifications to it and called the resulting protocol the ISO-R2 protocol, given in Figure 6.

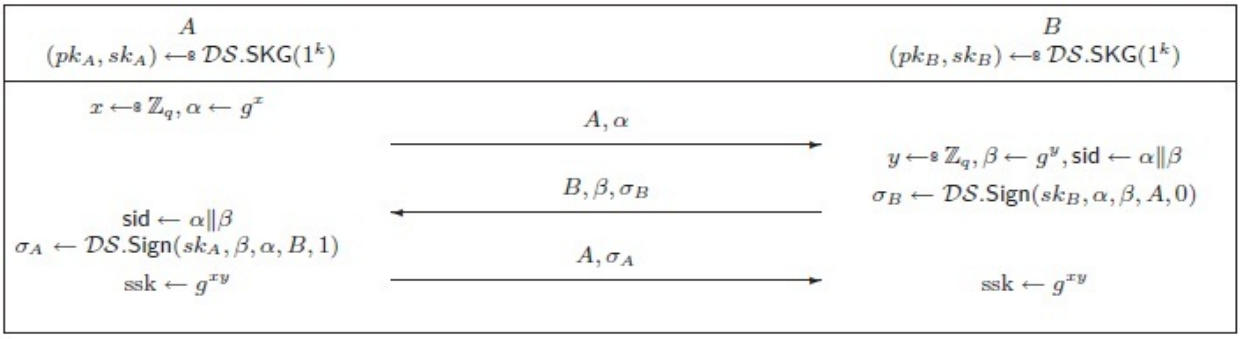


Figure 6: The ISO-R2 Protocol (Screenshot from page 15 [18]

Modification # 1: A role indicator ('0' for responder and '1' for initiator) is added into the message signed by each user. This is to prevent the following interleaving attack when the session ID is defined as the concatenation of the random group elements α and β sent by the intiator and the responder.

1. The adversary D corrupts A and then activates a new session, say Session i, between A (initiator) and B (responder).
2. After receiving the message (A, α) from A, D activates another session, say Session j, between B (initiator) and A (initiator).
3. After receiving the message (B, β) from B in Session j, D sends $(A, \alpha, \mathcal{DS}.\mathrm{Sign}(sk_A, \beta, \alpha, B))$ back to B in Session j.
4. B then responds to D with $(B, \mathcal{DS}.\mathrm{Sign}(sk_B, \alpha, \beta, A))$ and accepts the Session j with $sid = \beta||\alpha$.
5. In Session i, D sends $(B, \beta, \mathcal{DS}.\mathrm{Sign}(sk_B, \alpha, \beta, A))$ to A.
6. A responds to D in Session i with $(A, \mathcal{DS}.\mathrm{Sign}(sk_A, \beta, \alpha, B))$ and accepts this session with $sid = \alpha||\beta$.

It can be seen that A and B agree with the same session key, but under different session IDs, which is a problem. One way to do this, as is done in ISO-R2, is to add a role indicator so that an adversary who initiates two sessions between the same users with different roles cannot copy messages from one session to the other.

Modification # 2: To prevent the Reset-2 attack described in Section 5.1, the signature scheme used is a deterministic one. This way, a user is prevented from signing two different messages with the same signing key.

The next theorem shows that ISO-R2 is a Reset-2 secure protocol.

Theorem 3: The ISO-R2 protocol is secure in the strong-corruption Reset-2 model if the signing algorithm $\mathcal{DS}$ is a uf-cma-secure deterministic digital signature scheme, and the DDH assumption holds in the underlying group.

Proof Sketch: By way of contradiction, we assume that there exists an adversary A who breaks the ISO-R2 protocol under Reset-2 attack. We then define a restricted adversary A_1, who simulates all the RAKE-2 queries honestly for A, except that if A invokes the Test query with an instance that has no partner instance, A_1 aborts without any output. Let E be the event that the instance input by A into the Test query has no partner instance. We show that $\Pr[E]$ is negligible, otherwise we can break $\mathcal{DS}$ under adaptive CMA. This proves that the difference between the advantages of A and A_1 in RAKE-2 is negligible. We then define a restricted adversary A_2 from A_1 that guesses the input of the Test query invoked by A_1. We create an equality relating the advantages of A_1 and A_2 in RAKE-2. Finally, we modify RAKE-2 to get a new game G^1. We then show that the advantage of A_2 in G^1 is similar to that of A_2 in RAKE-2, otherwise the DDH property is violated. Putting together all the inequalities, we arrive at a contradiction.

Proof: Suppose by way of contradiction that A is an adversary who breaks the ISO-R2 protocol with non-negligible probability. Define a restricted adversary A_1 from A against ISO-R2 in the game RAKE-2 as follows. A_1 honestly answers all the queries made by A using its own oracles, except that when A makes the Test query with an input instance that has no partner instance, A_1 aborts without any output. When A inputs a bit b' to the Finalize procedure, so does A_1 and aborts.

We say that a 'Forge' event occurs if in the game, A generates a message-signature pair (m^*, σ^*) such that

- there exists a user $I \in \mathcal{U}$ such that true $\rightarrow \mathcal{DS}.\text{Verify}(pk_I, m^*, \sigma^*)$,
- user I is not corrupted at the time A generates (m^*, σ^*), and
- user I has never generated a signature on M^*.

Let E denote the event that A invokes the Test query with an instance (U^*, i^*) that has no partner instance. If E happens, then (U^*, i^*) generates a session key with some user V^* that has no partnering instance with (U^*, i^*). Then A must have posed as V^* to make (U^*, i^*) accept and generate a session key. This implies that A forged V^*'s signature. So if event E occurs, event F also occurs. Hence $\Pr[\text{E}] \leq \Pr[\text{Forge}]$. On the other hand if E does not occur, then A_1 and A are exactly the same, so $\mathbf{Adv}^{\text{rake-2}}_{\text{ISO-R2},A}(n) - \mathbf{Adv}^{\text{rake-2}}_{\text{ISO-R2},A_1}(n) = 0 \leq \Pr[\text{E}]$. If E occurs, then $\mathbf{Adv}^{\text{rake-2}}_{\text{ISO-R2},A_1}(n) = -0.5$, so $\mathbf{Adv}^{\text{rake-2}}_{\text{ISO-R2},A}(n) - \mathbf{Adv}^{\text{rake-2}}_{\text{ISO-R2},A_1}(n) \leq 0 \leq \Pr[\text{E}]$. Combining these results, we have

$$\mathbf{Adv}^{\text{rake-2}}_{\text{ISO-R2},A}(n) - \mathbf{Adv}^{\text{rake-2}}_{\text{ISO-R2},A_1}(n) \leq \Pr[\text{Forge}]. \tag{7}$$

Claim 3.1: $\Pr[\text{Forge}]$ is negligible.

Proof: Suppose for contradiction that $\Pr[\text{Forge}]$ is non-negligible. We show that $\mathcal{DS}$ is not uf-cma secure. Given the adversary A as above, we construct a signature forger X as follows. X is given a public key pk where $(pk, sk) \leftarrow \mathcal{DS}.\text{SKG}(1^n)$, and has access to a signing oracle $\mathcal{DS}.\text{Sign}(sk, .)$. X's

goal is to generate a signature using sk that has not been generated by the signing oracle, and is verifiable by pk. X randomly selects a user $U \in \mathcal{U}$ and assigns $pk_U = pk$. Then X generates all the long-lived keys for all the users in $\mathcal{U}\backslash\{U\}$. Now X simulates RAKE-2 for A. If a Forge event occurs in the simulation and $I = U$, then X outputs this forgery by A and aborts. Let m be the total number of users in the network, so $\Pr[I = U]$=$1/m$. Thus

$$\begin{aligned}\mathbf{Adv}^{\text{uf-cma}}_{\mathcal{DS},F}(n) &\geq \Pr[\text{Forge}] \times \Pr[I = U] \\ &= \frac{1}{m}\Pr[\text{Forge}].\end{aligned}$$

This implies that $\mathbf{Adv}^{\text{uf-cma}}_{\mathcal{DS},F}(n)$ is non-negligible, a contradiction. □

Now Pr[Forge] is negligible, so the difference between the advantages of A and A_1 in RAKE-2 are negligible. By the inequality derived in Claim 3.1 and (7), we have

$$\mathbf{Adv}^{\text{rake-2}}_{\text{ISO-R2},A_1}(n) \geq \mathbf{Adv}^{\text{rake-2}}_{\text{ISO-R2},A}(n) - m\mathbf{Adv}^{\text{uf-cma}}_{\mathcal{DS},F}(n). \tag{8}$$

Now we construct a restricted adversary A_2 from A_1, which outputs two integers l and l' after the Initialize phase. It simulates RAKE-2 for A_1 and answers all its queries honestly using its own oracles, except that if A_1 invokes the Test query with a session not between the l-th and l'-th instances, A_2 inputs a random bit to the Finalize procedure and aborts. Let q be the total number of new instance queries made by A_1. Then by analysis exactly similar to the one in Theorem 2 Case 1, we have

$$\mathbf{Adv}^{\text{rake-2}}_{\text{ISO-R2},A_2}(n) \geq \frac{1}{q(q-1)}\mathbf{Adv}^{\text{rake-2}}_{\text{ISO-R2},A_1}(n). \tag{9}$$

Now we modify RAKE-2 to obtain a new game G^1 as follows. In G^1, the simulator picks a random key (i.e. a random element in the underlying group), and sets it as the session key of the l'th and l'-th instances.

Claim 3.2: The difference between A_2's advantages in RAKE-2 and G^1 is negligible.

Proof: We show that if this difference is non-negligible, then we can construct an adversary D who breaks the DDH assumption.

Let D be an adversary against the DDH assumption. D is given a tuple $\{g, X = g^a, Y = g^b, Z\}$ and D's goal is to guess whether $Z = g^{ab}$ or Z is a random group element. D honestly simulates RAKE-2 for A_2 except that D simulates the l-th instance by settings the ephemeral public key as X and simulates the l'-th instance by setting the ephemeral public key as Y. D also sets Z as the session key of the l-th and l'-th instances. If A_2 wins the game, D outputs 1 and aborts. Otherwise D aborts without any input. Then

$$\begin{aligned}\mathbf{Adv}^{\text{DDH}}_{D}(n) &= \Pr[A_2 \text{ wins the game } |Z = g^{ab}] - \Pr[A_2 \text{ wins the game } |Z = g^r] \\ &= \Pr[\text{RAKE-2}_{\text{ISO-R2},A_2}(n) \to \text{true}] - \Pr[G^1_{\text{ISO-R2},A_2}(n) \to \text{true}] \\ &= \mathbf{Adv}^{\text{rake-2}}_{\text{ISO-R2},A_2}(n) - \mathbf{Adv}^{G^1}_{\text{ISO-R2},A_2}(n)\end{aligned}$$

This implies that if the difference between A_2's advantage in RAKE-2 and in G^1 is non-negligible, then $\mathbf{Adv}^{\text{DDH}}_{D}(n)$ is non-negligible, which is a contradiction. □

Finally, note that in G^1, A_2 can win the game only by guessing randomly, because the two choices for the session key are equally random. Hence $\mathbf{Adv}^{G^1}_{\text{ISO-R2},A_2}(n) = 0$. Combining this with (8), (9) and Claim 3.2, we get

$$\mathbf{Adv}^{\text{rake-2}}_{\text{ISO-R2},A}(n) \leq m\mathbf{Adv}^{\text{uf-cma}}_{\mathcal{DS},F}(n) + q(q-1)\mathbf{Adv}^{\text{DDH}}_{D}(n).$$

Since the quantity on LHS is non-negligible, at least one of the Adv terms on RHS is non-negligible, a contradiction. This completes the proof of Theorem 3. □

We can now apply the transformation to ISO-R2 to obtain a Reset-1 and Reset-2 secure protocol ISO-R, given the security of the digital signature scheme and the DDH assumption. The transformed protocol is shown in Figure 7. It can be seen that a is chosen in the initialization phase as part of A's secret long-lived key. During the protocol execution, when A chooses the random coins $\tilde{x}$, A re-randomizes them by passing them through the function F_a to obtain new random coins x which are used throughout the protocol.

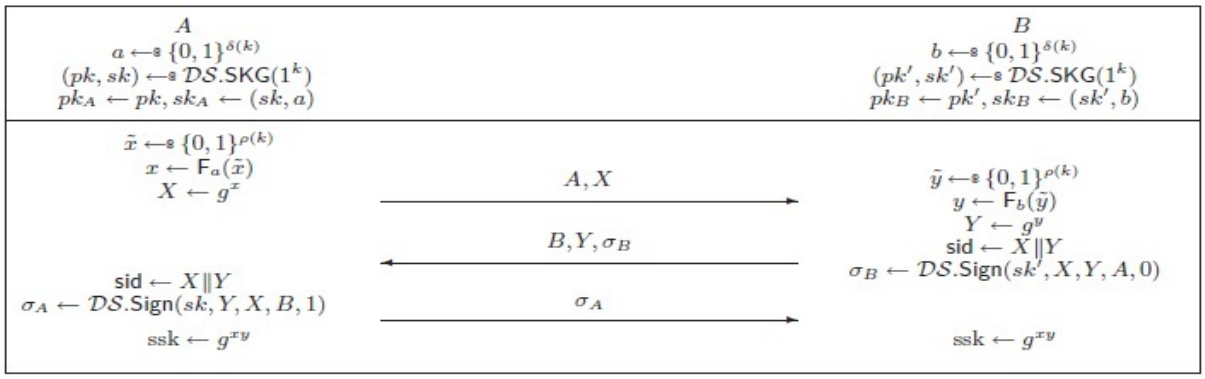


Figure 7: The ISO-R Protocol (Screenshot from page 18 [18]

7.2 A SKEME-based Reset-1 and Reset-2 Secure Protocol

The SKEME protocol is given in Figure 8. It uses an encryption scheme and a message authentication code scheme to achieve authentication and data integrity.

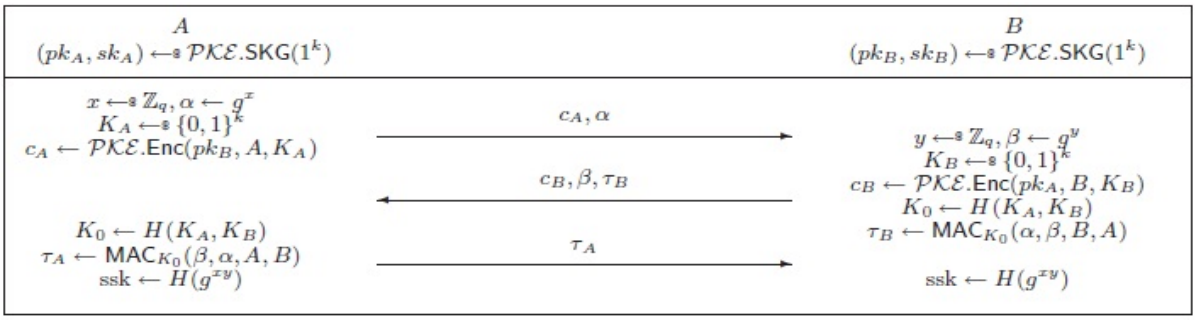


Figure 8: The SKEME Protocol (Screenshot from page 19 [18]

Since H is a public hash function, this protocol is Reset-1 insecure because the adversary knows the values of x and y and can trivially compute $H(g^{xy})$. It is also insecure in the strong-corruption Reset-2 model. The attack is given below.

- The adversary D activates user A to start a new session with user B.
- D relays the message c_A, α from A to B.
- Upon receiving the message c_B, β, τ_B from B, D makes a RevealState query to B and obtains the key K_0.
- D generates $\beta' \leftarrow g^{y'}$ where y' is chosen by the adversary, $\tau'_B \leftarrow \text{MAC}_{K_0}(\alpha, \beta', B, A)$, and sends a message (c_B, β', τ'_B) to A.
- Since τ'_B is generated using the correct key K_0, A accepts the session and outputs the session key $H(g^{xy})$.

In this attack, the RevealState query allows D to successfully pose as B and this makes the protocol insecure. Yang et al. proposed the PKEDH-R2 protocol (Figure 9) that is a modified version of SKEME. Here, a role indicator ('0' for responder, '1' for initiator) is added to the message being MAC-ed and the keying system for MAC is changed. Theorem 4 shows its security in the weak-corruption Reset-2 model under certain assumptions.

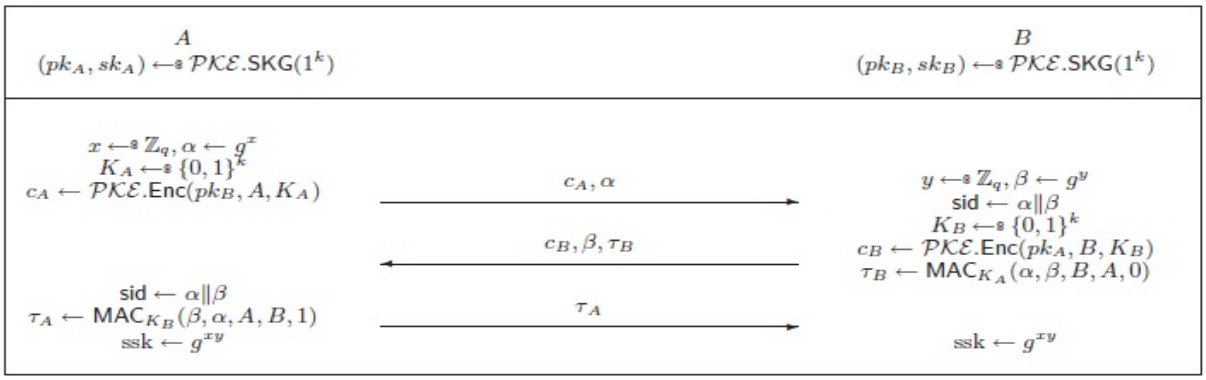


Figure 9: The PKEDH-R2 Protocol (Screenshot from page 2- [18]

Theorem 4: The PKEDH-R2 protocol is secure in the weak-corruption Reset-2 model if $\mathcal{PKE}$ is adaptive CCA secure, $\mathcal{MAC}$ is adaptive CMA secure, and the DDH assumption holds in the underlying group.

Proof Sketch: By way of contradiction, we assume that there exists an adversary A who breaks the PKEDH-R2 protocol under Reset-2 attack. We then define a restricted adversary A_1, who simulates all the RAKE-2 queries honestly for A, except that if A invokes the Test query with an instance that has no partner instance, A_1 aborts without any output. Let E be the event that the instance input by A into the Test query has no partner instance. We show that $\Pr[E]$ is negligible, otherwise we can break $\mathcal{PKE}$ under adaptive CCA or $\mathcal{MAC}$ under adaptive CMA. This in turn proves that the difference between the advantages of A and A_1 in RAKE-2 is negligible. We then define a restricted adversary A_2 from A_1 that guesses the input of the Test query invoked by A_1. Similar to the proof

of Theorem 3, we create an inequality relating the advantages of A_1 and A_2 in RAKE-2. Finally, we modify RAKE-2 to get a new game G^1. We then show that the advantage of A_2 in G^1 is similar to that of A_2 in RAKE-2, otherwise the DDH property is violated. Putting together all the inequalities, we arrive at a contradiction.

Proof: Before proceeding with the proof, we define an Encryption Aided Forger F. Let $(pk, sk) \leftarrow \mathcal{PKE}.\text{SKG}(1^n)$, and $c^* \leftarrow \mathcal{PKE}.\text{Enc}(pk, S, N^*)$, where S is a strong chosen by F, and N^* is randomly selected from the key space of $\mathcal{MAC}$ and unknown to F. F is given pk, c^* and has access to two oracles: a decryption oracle $\mathcal{O}_{sk}(.)$ which decrypts ciphertexts different from c^*, and an $\mathcal{O}_{N^*}(.)$ oracle which on input m returns $\text{MAC}_{N^*}(m)$. F's goal is to output m^*, $\text{MAC}_{N^*}(m^*)$ where F has never queried the oracle $\mathcal{O}_{N^*}(.)$ on message m^*.

Suppose by way of contradiction that A is an adversary who breaks the PKEDH-R2 protocol with high probability. Define a restricted adversary A_1 from A against PKEDH-R2 in the game RAKE-2 as follows. A_1 honestly answers all the queries made by A using its own oracles, except that when A makes the Test query with an input instance that has no partner instance, A_1 aborts without any output. When A inputs a bit b' to the Finalize procedure, so does A_1 and then it aborts. Let E be the event that the instance input by A into the Test query has no partner instance. If E occurs, we construct an Encryption Aided Forger F as follows.

F randomly selects two parties $U^*, V^* \in \mathcal{U}$, and generates all the long-lived keys for other users in the set $\mathcal{U} \backslash \{V^*\}$. Let q denote the maximum number of NewInstance queries made by A. F also selects an integer l randomly from the set $\{1, ..., q\}$. Then F asks its challenger with input $S = U^*$ and gets back the challenge $pk, c^* = \mathcal{PKE}.\text{Enc}(pk, U^*, N^*)$. F sets $pk_{V^*} = pk$. F's goal is to make A output $\text{MAC}_{N^*}(...)$. To do that, F simulates the game RAKE-2 for A honestly, except that

- if A does not make a Test query with an instance of U^*, F aborts without any output,
- if $pid_{U^*}^{i^*} \neq V^*$, then F aborts without any output (this and the previous item ensure that the Test session is between the i^*-th instance of U^* and user V^*),
- if (U^*, i^*) is not the l-th instance, G aborts without any output,
- if A makes a Corrupt query on V^*, F aborts without any output (this is because V^*'s secret key is sk, which is known only to F's challenger and not to F),
- for the l-th instance (i.e. (U^*, i^*)), F sets $c_{U^*} = c^*$, generates the ephemeral public and secret key pair (α, x) or (β, y), uses sk_{U^*} to get $(pk_{U^*}, V^*, N) \leftarrow \mathcal{PKE}.\text{Dec}(sk_{U^*}, c_{V^*})$ and then generates the tag τ_{U^*} honestly using N (note that N is K_{V^*} and N^* is K_{U^*}),
- if A sends a message $(c, ...)$ to V^* where $c \neq c^*$, F makes a query to its decryption oracle $\mathcal{O}_{sk}(.)$ on input c, and proceeds normally after getting back the response from $\mathcal{O}_{sk}(.)$,
- if A sends a message $(c^*, ...)$ to V^*, F queries its oracle $\mathcal{O}_{N^*}(.)$ to generate a response tag (i.e. a MAC with key N^*),
- when A sends a tag $\text{MAC}_{N^*}(...)$ to (U^*, i^*), the message MAC-ed here is different from all the messages MAC-ed previously because of the different role indicator. F outputs this MAC tag and the corresponding message as a successful forgery and aborts.

Let m be the total number of users in the network. Then

$$\begin{aligned}\Pr[F\text{succeeds}] &\geq \Pr[E] \times \ \Pr[\text{choosing } U^*, V^*] \times \ \Pr[\text{choosing } l\text{-th instance}] \\ &= \frac{1}{m(m-1)q}\Pr[E] \end{aligned} \tag{10}$$

Let ϵ =Pr[F succeeds]. Now given an Encryption Aided Forger F, we construct another adversary D against $\mathcal{PKE}$ in the IND-CCA game. D is given a public key pk and has access to a decryption oracle. D runs F as follows. When F asks for a challenge with input S, D randomly selects two numbers N_0 and N_1, and asks its challenger with input $S||N_0$ and $S||N_1$. After getting back the challenge c^*, D's goal is to determine whether c^* is the encryption of $S||N_0$ or $S||N_1$. D sets F's challenge to be pk, c^*. When F makes a MAC query on a message m, D returns $\text{MAC}_{N_0}(m)$ to F. Finally, if F successfully makes a forgery $\text{MAC}_{N_0}(m^*)$, then D outputs 0, indicating that c^* is the encryption of $S||N_0$. Otherwise, if F fails to produce a forgery, D outputs 1, signifying that c^* is the encryption of $S||N_1$. Let b be the bit output by D. Note that Pr[$b = 0$]=Pr[$b = 1$]=0.5. So we have

$$\begin{aligned}\mathbf{Adv}^{\text{cca}}_{\mathcal{PKE},D}(n) &= \Pr[D \text{ outputs } 0 \ |b = 0]\Pr[b = 0] + \Pr[D \text{ outputs } 1 \ |b = 1]\Pr[b = 1] - \frac{1}{2} \\ &= \frac{1}{2}\Pr[F \text{ succeeds } |b = 0] + \frac{1}{2}\Pr[F \text{ fails } |b = 1] - \frac{1}{2} \\ &= \frac{1}{2}\Pr[F \text{ succeeds } |b = 0] + \frac{1}{2}(1 - \Pr[F \text{ succeeds } |b = 1]) - \frac{1}{2} \\ &= \frac{1}{2}(\Pr[F \text{ succeeds } |b = 0] - \Pr[F \text{ succeeds } |b = 1]) \\ &= \frac{1}{2}(\epsilon - \mathbf{Adv}^{\text{cma}}_{\mathcal{MAC},F}(n)) \end{aligned} \tag{11}$$

The last equality comes from the fact that when $b = 0$, F is in the Encryption Aided Forger game, and when $b = 1$, c^* is independent of N_0 and F is in the normal chosen message attack game. Combining (10) and (11), we get

$$\Pr[E] \leq m(m-1)q(2\mathbf{Adv}^{\text{cca}}_{\mathcal{PKE},D}(n) + \mathbf{Adv}^{\text{cma}}_{\mathcal{MAC},F}(n). \tag{12}$$

This equation implies that if Pr[E] is non-negligible, one can break $\mathcal{PKE}$ or $\mathcal{MAC}$, a contradiction. Therefore, we can assume with high probability that E does not occur, i.e., A invokes the Test query with an instance that has a partner instance.

Now, if E does not happen, A_1 and A are the same, so we have

$$\mathbf{Adv}^{\text{rake-2}}_{\text{ISO-R2},A}(n) - \mathbf{Adv}^{\text{rake-2}}_{\text{ISO-R2},A_1}(n) \leq \Pr[E].$$

Combining this with (12), we get

$$\mathbf{Adv}^{\text{rake-2}}_{\text{ISO-R2},A}(n) - \mathbf{Adv}^{\text{rake-2}}_{\text{ISO-R2},A_1}(n) \leq m(m-1)q(2\mathbf{Adv}^{\text{cca}}_{\mathcal{PKE},D}(n) + \mathbf{Adv}^{\text{cma}}_{\mathcal{MAC},F}(n)). \tag{13}$$

We now proceed in exactly the same fashion as in the proof of Theorem 3. Given A_1, we define another restricted adversary A_2 which outputs two integers l and l'. Then we get

$$\mathbf{Adv}^{\text{rake-2}}_{\text{ISO-R2},A_2}(n) \geq \frac{1}{q(q-1)}\mathbf{Adv}^{\text{rake-2}}_{\text{ISO-R2},A_1}(n). \tag{14}$$

Now we modify RAKE-2 to obtain a new game G^1, in which the simulator picks a random element of the underlying subgroup and sets it as the session key of the l'th and l'-th instances. Then by the second claim in the proof of Theorem 3, we get

$$\mathbf{Adv}^{\text{DDH}}_{D}(n) = \mathbf{Adv}^{\text{rake-2}}_{\text{ISO-R2},A_2}(n) - \mathbf{Adv}^{G^1}_{\text{ISO-R2},A_2}(n) \tag{15}$$

Finally, A_2 can win G^1 only by guessing randomly, because the two choices for the session key are equally random. Hence $\mathbf{Adv}^{G^1}_{\text{ISO-R2},A_2}(n) = 0$. We now combine (13), (14) and (15).

$$\begin{aligned}
\mathbf{Adv}^{\text{rake-2}}_{\text{ISO-R2},A}(n) &\leq m(m-1)q(2\mathbf{Adv}^{\text{cca}}_{\mathcal{PKE},D}(n) + \mathbf{Adv}^{\text{cma}}_{\mathcal{MAC},F}(n)) + \mathbf{Adv}^{\text{rake-2}}_{\text{ISO-R2},A_1}(n) \\
&\leq m(m-1)q(2\mathbf{Adv}^{\text{cca}}_{\mathcal{PKE},D}(n) + \mathbf{Adv}^{\text{cma}}_{\mathcal{MAC},F}(n)) + q(q-1)\mathbf{Adv}^{\text{rake-2}}_{\text{ISO-R2},A_2}(n) \\
&\leq m(m-1)q(2\mathbf{Adv}^{\text{cca}}_{\mathcal{PKE},D}(n) + \mathbf{Adv}^{\text{cma}}_{\mathcal{MAC},F}(n)) + q(q-1)\mathbf{Adv}^{\text{DDH}}_{D}(n)
\end{aligned}$$

Since the quantity on LHS is non-negligible by assumption, at least one of the three Adv terms on RHS is non-negligible, a contradiction. This completes the proof of Theorem 4. □

We can now apply the transformation to PKEDH-R2 to obtain a Reset-1 and Reset-2 secure protocol PKEDH-R, given the security of the encryption scheme, message authentication code scheme and the DDH assumption. The PKEDH-R protocol is shown in Figure 10. It can be seen that a is chosen in the initialization phase as part of A's secret long-lived key. During the protocol execution, when A chooses the random coins $\tilde{x}$, A re-randomizes them by passing them through the function F_a to obtain new random coins x, K_A, r which are then used throughout the protocol.

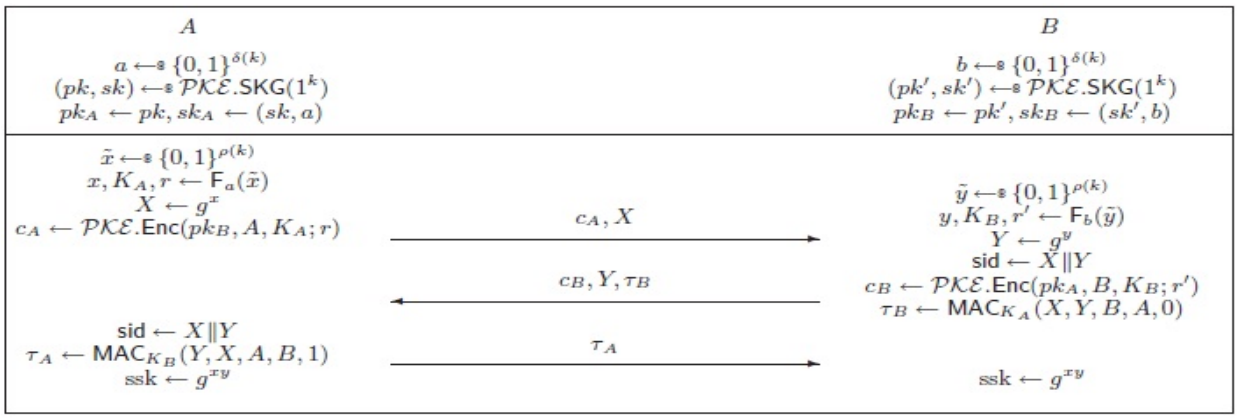


Figure 10: The PKEDH-R Protocol (Screenshot from page 21 [18]

8 Conclusion

In this report, we studied in complete detail the research paper published by Yang et al. [18] in 2011. This paper initiates the formal study on Authenticated Key Exchange under bad randomness. Two possible scenarios are considered, where the randomness of a protocol is directly or indirectly

controlled by the adversary, causing the session keys or the secret keys of participants to be leaked. Under both the scenarios, it is shown that many widely known AKE protocols are insecure. Finally, a way to transform an insecure protocol to a secure one is described, followed by concrete examples. Our contribution to Yang et al.'s work is that we have implemented an attack on HMQV protocol given by Menezes and Ustaoglu [16]. In addition, we give the proofs and the computational/logical details that have been omitted in the original work.

References

[1] Entity authentication mechanisms - Part 3: Entity authentication using asymmetric techniques. *ISO/IEC IS 9798-3*, 1993.

[2] Digital signature standard (DSS). *NIST FIPS PUB 186-3*, Jun 2009.

[3] W. Aiello, S.M. Bellovin, M. Blaze, R. Canetti, J. Ioannidis, A.D. Keromytis, and O. Reingold. Just fast keying: Key agreement in a hostile internet. *ACM Trans. Inf. Syst. Secur.*, 7(2):242–273, 2004.

[4] M. Bellare, M. Fischlin, S. Goldwasser, and S. Micali. Identification protocols secure against reset attacks. *EUROCRYPT 2001*, pages 495–511.

[5] R. Canetti and H. Krawczyk. Analysis of key exchange protocols and their use for building secure channels. *EUROCRYPT 2001*, pages 453–474.

[6] R. Canetti and H. Krawczyk. Security analysis of IKE's signature-based key-exchange protocol. *CRYPTO 2002*, pages 143–161.

[7] A. Desai, A. Hevia, and Y. L. Yin. A practice-oriented treatment of pseudorandom number generators. *EUROCRYPT 2002*, pages 368–383.

[8] R. Diestel. *Graph Theory.* Springer-Verlag Heidelberg, New York, third edition, 2005.

[9] Y. Dodis. Exposure-resilient cryptography. *PhD Thesis, MIT*, 2000.

[10] D. Eastlake, S. Crocker, and J. Schiller. *IETF RFC 1750: Randomness Recommendations for Security*, 1994.

[11] A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. *CRYPTO'86*, pages 186–194.

[12] H. Krawczyk. HMQV: A high-performance secure Diffie-Hellman protocol. *CRYPTO 2005*, pages 546–566.

[13] H. Krawczyk. SIGMA: The 'SIGn-and-MAc' approach to authenticated Diffie-Hellman and its use in the IKE protocols. *CRYPTO 2003*, pages 400–425.

[14] H. Krawczyk. SKEME: A versatile secure key exchange mechanism for internet. *NDSS*, pages 114–127, 1996.

[15] T. Matthews. Suggestions for random number generation in software. *RSA Laboratories Bulletin # 1*, January 1996.

[16] A. Menezes and B. Ustaoglu. On reusing ephemeral keys in Diffie-Hellman key agreement protocols. *IJACT*, 2(2):154–158, 2010.

[17] T. Ristenpart and S. Yilek. When good randomness goes bad: Virtual machine reset vulnerabilities and hedging deployed cryptography. *Network and Distributed System Security Symposium*, 2010.

[18] G. Yang, S. Duan, D.S. Wong, C.H. Tan, and H. Wang. Authenticated key exchange under bad randomness. *Proceedings of the 15th international conference on Financial Cryptography and Data Security*, pages 113–126, 2011.