Unknown Key-Share Attacks on the Station-to-Station (STS) Protocol *

Simon Blake-Wilson
Certicom Corp.
sblakewi@certicom.com

Alfred Menezes
University of Waterloo
ajmeneze@cacr.math.uwaterloo.ca

Abstract

This paper presents some new unknown key-share attacks on STS-MAC, the version of the STS key agreement protocol which uses a MAC algorithm to provide key confirmation. Various methods are considered for preventing the attacks.

1 Introduction

Key establishment is the process by which two (or more) entities establish a shared secret key. The key may subsequently be used to achieve some cryptographic goal, such as confidentiality or data integrity. Ideally, the established key should have precisely the same attributes as a key established face-to-face — for example, it should be shared by the (two) specified entities, it should be distributed uniformly at random from the key space, and no unauthorized (and computationally bounded) entity should learn anything about the key.

Key establishment protocols come in various flavors. In *key transport* protocols, a key is created by one entity and securely transmitted to the second entity, while in *key agreement* protocols both parties contribute information which is used to derive the shared secret key. In *symmetric* protocols the two entities a priori possess common secret information, while in *asymmetric* protocols the two entities share only public information that has been authenticated. This paper is concerned with two-party key agreement protocols in the asymmetric setting.

Unfortunately, the requirement that key agreement protocols have the same properties as face-to-face key establishment is too vague to be much help to protocol designers, who instead focus on designing protocols to meet more explicit requirements. Implicit key authentication and key confirmation are two explicit requirements that are often considered essential.

Let A and B be two honest entities, i.e., legitimate entities who execute the steps of a protocol correctly. Informally speaking, a key agreement protocol is said to provide *implicit key authentication* (of B to A) if entity A is assured that no other entity aside from a specifically identified second entity B can possibly learn the value of a particular secret key. Note that the property of implicit key authentication does not necessarily mean that A is assured of B actually

*Originally published in September 1998; Revised in November 1998.

possessing the key. A key agreement protocol which provides implicit key authentication to both participating entities is called an *authenticated key agreement (AK)* protocol.

Informally speaking, a key agreement protocol is said to provide *explicit key confirmation* (of B to A) if entity A is assured that the second entity B *has actually* computed the agreed key. The protocol provides *implicit key confirmation* if A is assured that B *can* compute the agreed key. While explicit key confirmation appears to provide stronger assurances to A than implicit key confirmation (in particular, the former implies the latter), it appears that, for all practical purposes, the assurances are in fact the same. That is, the assurance that A requires in practice is merely that B can compute the key rather than that B has actually computed the key. Indeed in practice, even if a protocol does provide explicit key confirmation, it cannot guarantee to A that B will not lose the key between key establishment and key use. Thus it would indeed seem that implicit key confirmation and explicit key confirmation are in practice very similar.

If both implicit key authentication and (implicit or explicit) key confirmation (of B to A) are provided, then the key establishment protocol is said to provide *explicit key authentication* (of B to A). A key agreement protocol which provides explicit key authentication to both participating entities is called an *authenticated key agreement with key confirmation (AKC)* protocol.

In addition to implicit key authentication and key confirmation, a number of other desirable security attributes of key agreement protocols have been identified including known-key security, forward secrecy, key-compromise impersonation, and unknown key-share. These are typically properties possessed by face-to-face key establishment which may be more or less important when a key establishment protocol is used to provide security in real-life applications.

An *unknown key-share* (UKS) attack on an AK or AKC protocol is an attack whereby an entity A ends up believing she shares a key with B, and although this is in fact the case, B mistakenly believes the key is instead shared with an entity $E \neq A$. The significance of UKS attacks on AK and AKC protocols is further discussed in §3.

This paper presents some new on-line UKS attacks on STS-MAC, the variant of the station-to-station (STS) [11] AKC protocol which uses a MAC to provide key confirmation. For an extensive survey on key establishment, see Chapter 12 of [25]. For a recent survey on authenticated Diffie-Hellman key agreement protocols, see [10]. Formal definitions of authenticated key agreement can be found for the symmetric setting in [7] and for the asymmetric setting in [9].

The remainder of this paper is organized as follows. The STS protocol is described in §2. In §3 we present the new on-line UKS attacks on STS-MAC, and consider ways of preventing the attacks. In §4, we examine the plausibility of an assumption regarding signature schemes that is required in order for the attacks to succeed. §5 makes concluding remarks.

2 Description of STS

The station-to-station (STS) protocol [11] is a Diffie-Hellman-based AKC protocol that purports to provide both (mutual) implicit key authentication and (mutual) key confirmation, and additionally appears to possess desirable security attributes such as forward secrecy and key-compromise impersonation. The STS protocol, as described in [11], provides (explicit) key confirmation by using the agreed key K in a symmetric-key encryption scheme; we call this protocol STS-ENC. A variant of STS mentioned in [11], which we call STS-MAC, provides (explicit) key confirmation by using the agreed key K in a MAC algorithm.

STS-MAC may be preferred over STS-ENC in many practical scenarios because of existing export or usage restrictions on secure encryption. Moreover, the use of encryption to provide key confirmation in STS-ENC is questionable — traditionally the sole goal of encryption is to provide confidentiality and if an encryption scheme is used to demonstrate possession of a key then it is shown by decryption, not by encryption. One advantage of STS-ENC over STS-MAC is that the former can facilitate the provision of anonymity.

Many protocols related to STS have appeared in the literature (e.g., [5], [14], [18]). It should be noted, however, that these protocols cannot be considered to be minor variants of STS — as this paper shows, the former protocols have some security attributes that are lacking in STS.

The following notation is used throughout the paper.

Notation

A, B	Honest entities.
E	The adversary.
S_A	A's (private) signing key for a signature scheme S.
P_A	A's (public) verification key for S.
$S_A(M)$	A's signature on a message M.
Cert_A	A's certificate containing A's identifying information, A's public signature key P_A, and possibly some other information.
$E_K(M)$	Encryption of M using a symmetric-key encryption scheme with key K.
$\text{MAC}_K(M)$	Message authentication code of M under key K.
G, α, n	Diffie-Hellman parameters; α is an element of prime order n in the finite multiplicative group G.
r_A	A's ephemeral Diffie-Hellman private key; $1 \leq r_A \leq n-1$.
K	Ephemeral Diffie-Hellman shared secret; $K = \alpha^{r_A r_B}$.

The two STS variants are presented below (see also [11, 25, 32]). In both descriptions, A is called the *initiator*, while B is called the *responder*.

STS-MAC protocol.

The STS-MAC protocol is depicted below. Initiator A selects a random secret integer r_A, $1 \leq r_A \leq n-1$, and sends to B the message (1). Upon receiving (1), B selects a random secret integer r_B, $1 \leq r_B \leq n-1$, computes the shared secret $K = (\alpha^{r_A})^{r_B}$, and sends message (2) to A. Upon receiving (2), A uses Cert_B to verify the authenticity of B's signing key P_B, verifies B's signature on the message $(\alpha^{r_B}, \alpha^{r_A})$, computes the shared secret $K = (\alpha^{r_B})^{r_A}$, and verifies the MAC on $S_B(\alpha^{r_B}, \alpha^{r_A})$. A then sends message (3) to B. Upon receipt of (3), B uses Cert_A to verify the authenticity of A's signing key P_A, verifies A's signature on the message $(\alpha^{r_A}, \alpha^{r_B})$, and verifies the MAC on $S_A(\alpha^{r_A}, \alpha^{r_B})$. If at any stage a check or verification performed by A or B fails, then that entity terminates the protocol run, and rejects.

(1)	$A \to B$	$A,\ \alpha^{r_A}$
(2)	$A \leftarrow B$	$\text{Cert}_B,\ \alpha^{r_B},\ S_B(\alpha^{r_B}, \alpha^{r_A}),\ \text{MAC}_K(S_B(\alpha^{r_B}, \alpha^{r_A}))$
(3)	$A \to B$	$\text{Cert}_A,\ S_A(\alpha^{r_A}, \alpha^{r_B}),\ \text{MAC}_K(S_A(\alpha^{r_A}, \alpha^{r_B}))$

STS-ENC protocol.

The STS-ENC protocol is given below. For the sake of brevity, the checks that should be performed by A and B are henceforth omitted.

(1)	$A \to B$	$A,\ \alpha^{r_A}$
(2)	$A \leftarrow B$	$\text{Cert}_B,\ \alpha^{r_B},\ E_K(S_B(\alpha^{r_B}, \alpha^{r_A}))$
(3)	$A \to B$	$\text{Cert}_A,\ E_K(S_A(\alpha^{r_A}, \alpha^{r_B}))$

3 Unknown key-share attacks

An *unknown key-share* (UKS) attack on a key agreement protocol is an attack whereby an entity A ends up believing she shares a key with B, and although this is in fact the case, B mistakenly believes the key is instead shared with an entity $E \neq A$. In this scenario, we say that B has been led to false beliefs. If B is the protocol's initiator, then the attack is called a UKS attack *against the initiator*. Otherwise, the attack is called a UKS attack *against the responder*.

It is important to note that if an AK or AKC protocol succumbs to a UKS attack in which E is a dishonest entity (this is the case with the attacks presented in this paper), then this does not contradict the implicit key authentication property of the protocol — by definition, the provision of implicit key authentication is only considered in the case where B engages in the protocol with an honest entity (which E isn't).

AN ATTACK SCENARIO. A hypothetical scenario where a UKS attack can have damaging consequences is the following; this scenario was first described in [11]. Suppose that B is a bank branch and A is an account holder. Certificates are issued by the bank headquarters and within each certificate is the account information of the holder. Suppose that the protocol for electronic deposit of funds is to exchange a key with a bank branch via an AKC protocol. At the conclusion of the protocol run, encrypted funds are deposited to the account number in the certificate. Suppose that no further authentication is done in the encrypted deposit message (which might be the case to save bandwidth). If the UKS attack mentioned above is successfully launched then the deposit will be made to E's account instead of A's account.

ANOTHER ATTACK SCENARIO. Another scenario where a UKS attack can be damaging is the following. Suppose that B controls access to a suite of sensitive applications (e.g. salary databases). Each application has a password associated with it. The password is chosen and securely distributed by a CA to B and to all entities entitled to access that application. The CA also certifies public keys of all potential users of one (or more) of the applications. A user A gains access to an application by supplying to B the password that is specific to that application. This can be done securely as follows. When A wants to gain access to the application, she and B engage in a single run of an AKC protocol to establish shared keys K_1 and K_2 (K_1 and K_2 are derived from the shared secret established). A then authenticates and encrypts the password using the keys and sends the result to B. B checks the encrypted authenticated password and supplies access to A. Once access has been granted, the application establishes new keys with A to secure the subsequent use of the application.

If the AKC protocol does not provide unknown key-share, an active adversary E can induce B into believing that he shares the keys K_1 and K_2 with E, while A correctly believes that she shares the keys with B. E may then use the encrypted authenticated password sent by A to gain access to the application.

SIGNIFICANCE OF UKS ATTACKS. The importance of preventing UKS attacks has been debated in the literature. It is interesting to note that prevention of UKS attacks was one of the original design principles of STS [11]. Here we make two observations about the relevance of UKS attacks. First, notice that traditional, face-to-face key establishment is not susceptible to UKS attacks. Therefore anyone implementing a key establishment protocol that does not prevent UKS attacks as a drop-in replacement for face-to-face key establishment must check whether UKS attacks represent a security concern in the application. Second, notice that a UKS attack on an AKC protocol is more serious than a UKS attack on an AK protocol (which does not provide key confirmation). As stated in [9], keys established using AK protocols should be confirmed prior to cryptographic use. Indeed, some standards such as [4] take the conservative approach of mandating key confirmation of keys agreed in an AK protocol. If appropriate key confirmation is subsequently provided, then the attempt at a UKS attack will be detected. For this reason, the above hypothetical banking scenario (in particular, the assumption that no further authentication is performed after termination of the key agreement protocol) is realistic if an AKC protocol is used (since key confirmation has already been provided), and unrealistic if an AK protocol is used (since key confirmation has not yet been provided).

The remainder of this section discusses UKS attacks on STS-MAC and STS-ENC. §3.1 describes well-known public key substitution UKS attacks (for example, see [24, 25]). These attacks can be prevented if a CA checks possession of private keys during the certification process. §3.2 presents new on-line UKS attacks on STS-MAC that are not prevented simply by checking knowledge of private keys during certification. It suggests other methods which may be used to prevent the new attacks. The attacks are similar in spirit to Kaliski's recent attack [20] on the AK protocol of [21] — however the attacks we present are more damaging because, unlike Kaliski's attack, they are not prevented by appropriate key confirmation. Finally, in §3.3 we consider possible UKS attacks on STS-ENC which may not be prevented by checking knowledge of private keys during certification. The attacks in §3.3 are considerably more far-fetched than the attacks in §3.2, but they demonstrate the value of public-key validation and formal protocol analysis.

3.1 Public key substitution UKS attacks

This section describes well-known public key substitution UKS attacks on STS-MAC and STS-ENC.

Attack against the responder.

In this UKS attack against the responder, the adversary E registers A's public key P_A as its own; i.e., $P_E = P_A$. When A sends B message (1), E intercepts it and replaces the identity A with E. E then passes message (2) from B to A unchanged. Finally E intercepts message (3), and replaces Cert_A with Cert_E. Since $P_A = P_E$, we have $S_A(\alpha^{r_A}, \alpha^{r_B}) = S_E(\alpha^{r_A}, \alpha^{r_B})$. Hence B accepts the key K and believes that K is shared with E, while in fact it is shared with A. Note that E does not learn the value of K. The attack is depicted below. The notation $A \hookrightarrow B$ means that A transmitted a message intended for B, which was intercepted by the adversary and not delivered to B.

(1)	$A \hookrightarrow B$	$A,\ \alpha^{r_A}$
(1')	$E \rightarrow B$	$E,\ \alpha^{r_A}$
(2)	$E \leftarrow B$	$\mathrm{Cert}_B,\ \alpha^{r_B},\ S_B(\alpha^{r_B}, \alpha^{r_A}),\ \mathrm{MAC}_K(S_B(\alpha^{r_B}, \alpha^{r_A}))$
(2')	$A \leftarrow E$	$\mathrm{Cert}_B,\ \alpha^{r_B},\ S_B(\alpha^{r_B}, \alpha^{r_A}),\ \mathrm{MAC}_K(S_B(\alpha^{r_B}, \alpha^{r_A}))$
(3)	$A \hookrightarrow B$	$\mathrm{Cert}_A,\ S_A(\alpha^{r_A}, \alpha^{r_B}),\ \mathrm{MAC}_K(S_A(\alpha^{r_A}, \alpha^{r_B}))$
(3')	$E \rightarrow B$	$\mathrm{Cert}_E,\ S_A(\alpha^{r_A}, \alpha^{r_B}),\ \mathrm{MAC}_K(S_A(\alpha^{r_A}, \alpha^{r_B}))$

Attack against the initiator.

E can similarly launch a UKS attack against the initiator A by registering B's public P_B as its own. The attack is depicted below.

(1)	$A \rightarrow E$	$A,\ \alpha^{r_A}$
(1')	$E \rightarrow B$	$A,\ \alpha^{r_A}$
(2)	$A \hookleftarrow B$	$\mathrm{Cert}_B,\ \alpha^{r_B},\ S_B(\alpha^{r_B}, \alpha^{r_A}),\ \mathrm{MAC}_K(S_B(\alpha^{r_B}, \alpha^{r_A}))$
(2')	$A \leftarrow E$	$\mathrm{Cert}_E,\ \alpha^{r_B},\ S_B(\alpha^{r_B}, \alpha^{r_A}),\ \mathrm{MAC}_K(S_B(\alpha^{r_B}, \alpha^{r_A}))$
(3)	$A \rightarrow E$	$\mathrm{Cert}_A,\ S_A(\alpha^{r_A}, \alpha^{r_B}),\ \mathrm{MAC}_K(S_A(\alpha^{r_A}, \alpha^{r_B}))$
(3')	$E \rightarrow B$	$\mathrm{Cert}_A,\ S_A(\alpha^{r_A}, \alpha^{r_B}),\ \mathrm{MAC}_K(S_A(\alpha^{r_A}, \alpha^{r_B}))$

Preventing the attacks.

Both these public key substitution attacks are well-known and are usually prevented by requiring that entities prove to the certificate-issuing authority possession of the private keys corresponding to their public keys during the certification process. The attacks can also be launched against STS-ENC; it this case, an alternate way to prevent the attacks is to encrypt certificates using the shared key K.

3.2 On-line UKS attacks on STS-MAC

This section describes the new on-line UKS attacks on STS-MAC. The following assumptions are made in order for the attacks to be effective.

1. The signature scheme S used in STS has the following *duplicate-signature key selection* property. Suppose that P_A (A's public key) and A's signature s_A on a message M are known. Then the adversary is able to select a key pair (P_E, S_E) with respect to which s_A

is also E's signature on the message M. The plausibility of this assumption is examined in §4, where it is shown that the RSA, Rabin, ElGamal, DSA and ECDSA signature schemes all possess the duplicate-signature key selection property in certain situations.

2. E is able to get its public key certified during a run of the STS protocol. This assumption is plausible, for instance, in situations where delays in the transmission of messages are normal, and where the CA is on-line.

Attack against the responder.

This new UKS attack on STS-MAC is similar to the public key substitution attack against the responder in §3.1. After A sends message (3), E intercepts it and selects a key pair (P_E, S_E) for the employed signature scheme such that $S_E(\alpha^{r_A}, \alpha^{r_B}) = S_A(\alpha^{r_A}, \alpha^{r_B})$. E then obtains a certificate Cert_E for P_E, and transmits message (3') to B.

Attack against the initiator.

This new UKS attack on STS-MAC is similar to the public key substitution attack against the initiator in §3.1. After B sends message (2), E intercepts it and selects a key pair (P_E, S_E) for the employed signature scheme such that $S_E(\alpha^{r_B}, \alpha^{r_A}) = S_B(\alpha^{r_B}, \alpha^{r_A})$. E then obtains a certificate Cert_E for P_E, and transmits message (2') to A.

Preventing the attacks.

In the on-line UKS attacks, the adversary knows the private key S_E corresponding to its chosen public key P_E. Hence, unlike the case of the public key substitution attacks, the on-line attacks cannot be prevented by requiring that entities prove to the certificate-issuing authority possession of the private keys corresponding to their public keys during the certification process.

The following outlines some measures that can be taken to prevent the on-line UKS attacks on STS-MAC.

1. If A sends its certificate Cert_A in flow (1) rather than in flow (3), then the on-line UKS attack against the responder cannot be launched; however the on-line UKS attack against the initiator still succeeds.

2. If certificates are exchanged a priori, i.e., prior to the protocol run, then the on-line UKS attacks fail. A priori exchanges of certificates may be undesirable in practice because it increases the number of protocol flows.

3. Including the identities of the sender and intended receiver as well as the flow number[1] in the messages being signed prevents the on-line UKS attacks. Inclusion of the flow number and the identity of the message sender may help guard against attacks yet to be discovered. (See [26] for an example of how inclusion of flow numbers can help guard against

[1]In this paper, we assume that message fields such as flow numbers, identities, and group elements, are represented using fixed-length encodings and concatenated. Otherwise, some other unique prefix-free encoding such as ASN.1 DER [15, 16] should be used.

certain attacks on entity authentication mechanisms.) These modifications add negligible computational overhead to the protocol and follow the generic philosophy expounded in [9] and [21]. The revised protocol is shown below.

(1)	$A \to B$	$A,\ \alpha^{r_A}$
(2)	$A \leftarrow B$	$\mathrm{Cert}_B,\ \alpha^{r_B},\ S_B(2, B, A, \alpha^{r_B}, \alpha^{r_A})$, $\mathrm{MAC}_K(S_B(2, B, A, \alpha^{r_B}, \alpha^{r_A}))$
(3)	$A \to B$	$\mathrm{Cert}_A,\ S_A(3, A, B, \alpha^{r_A}, \alpha^{r_B}),\ \mathrm{MAC}_K(S_A(3, A, B, \alpha^{r_A}, \alpha^{r_B}))$

4. In the original STS-MAC protocol and the modification presented in item 3 above, the agreed key K is used as the MAC key for the purpose of providing *explicit* key confirmation. A passive adversary now has some information about K — the MAC of a known message under K. The adversary can use this to distinguish K from a key selected uniformly at random from the key space[2]. The elegant general principle that in the face of a computationally bounded adversary a computationally indistinguishable key can later be used in place of a traditional face-to-face secret key anywhere without sacrificing security can therefore not be applied (and security must be analyzed on a case-by-case basis). Another drawback of providing explicit key confirmation in this way is that the agreed key K may be subsequently used with a different cryptographic mechanism than the MAC algorithm — this violates a fundamental cryptographic principle that a key should not be used for more than one purpose.

   An improvement, therefore, is to provide *implicit*, rather than explicit, key confirmation. Two keys K and K' are derived from $\alpha^{r_A r_B}$ using a cryptographic hash function H. In practice, this can be achieved by setting $K \| K' = H(\alpha^{r_A r_B})$, or $K = H(01, \alpha^{r_A r_B})$ and $K' = H(10, \alpha^{r_A r_B})$. K' is used as the MAC key for the session, while K is used as the agreed session key. The revised protocol is depicted below.

(1)	$A \to B$	$A,\ \alpha^{r_A}$
(2)	$A \leftarrow B$	$\mathrm{Cert}_B,\ \alpha^{r_B},\ S_B(2, B, A, \alpha^{r_B}, \alpha^{r_A})$, $\mathrm{MAC}_{K'}(S_B(2, B, A, \alpha^{r_B}, \alpha^{r_A}))$
(3)	$A \to B$	$\mathrm{Cert}_A,\ S_A(3, A, B, \alpha^{r_A}, \alpha^{r_B}),\ \mathrm{MAC}_{K'}(S_A(3, A, B, \alpha^{r_A}, \alpha^{r_B}))$

   We imagine that this protocol (and also the protocol in item 6 below) can be analyzed by modelling the hash function H as a random oracle [6].

5. Instead of including the identities of the entities in the signed message, one could include them in the *key derivation function*, whose purpose is to derive the shared key from the shared secret $\alpha^{r_A r_B}$. In the protocol of item 3, the shared secret key would be $K = H(\alpha^{r_A r_B}, A, B)$, while in the 2 protocols of item 4, the shared keys would be (i) $K \| K' = H(\alpha^{r_A r_B}, A, B)$ and (ii) $K' = H(01, \alpha^{r_A r_B}, A, B)$ and $K = H(10, \alpha^{r_A r_B}, A, B)$.

   However, key derivation functions have not been well-studied by the cryptographic community. In particular, the desirable security properties of a key derivation function have not yet been specified. For this reason, the protocols presented in items 3 and 4 are preferred over the variants which include identities in the key derivation function.

6. The protocols in item 4 provide implicit key confirmation. While the assurance that the other entity has actually computed the shared key K is not provided, each entity does

[2]The key space here is $\mathcal{K} = \{\alpha^i \ : \ 1 \le i \le n-1\}$.

get the assurance that the other has computed the shared secret $\alpha^{r_A r_B}$. Implicit key confirmation is still provided (to a somewhat lesser degree) if the MACs are not included in the flows. The revised protocol is shown below:

(1)	$A \to B$	$A,\ \alpha^{r_A}$
(2)	$A \leftarrow B$	$\text{Cert}_B,\ \alpha^{r_B},\ S_B(2, B, A, \alpha^{r_B}, \alpha^{r_A})$
(3)	$A \to B$	$\text{Cert}_A,\ S_A(3, A, B, \alpha^{r_A}, \alpha^{r_B})$

7. ISO 11770-3 has one variant each of the STS-ENC and STS-MAC protocols — these are included as "Key agreement mechanism 7" in [18]. Both these variants resist the on-line UKS attacks. The ISO variant of STS-MAC, which we call ISO-STS-MAC, is the following:

(1)	$A \to B$	$A,\ \alpha^{r_A}$
(2)	$A \leftarrow B$	$\text{Cert}_B,\ \alpha^{r_B},\ S_B(\alpha^{r_B}, \alpha^{r_A}, A),\ \text{MAC}_K(\alpha^{r_B}, \alpha^{r_A}, A)$
(3)	$A \to B$	$\text{Cert}_A,\ S_A(\alpha^{r_A}, \alpha^{r_B}, B),\ \text{MAC}_K(\alpha^{r_A}, \alpha^{r_B}, B)$

Notice that, unlike the original description of STS-MAC, identities of the intended recipients are included in the signatures in ISO-STS-MAC. This was apparently done in order to be conformant with the entity authentication mechanisms in ISO 9798-3 [17], rather than because of a security concern with STS without the inclusion of identities. Another difference between ISO-STS-MAC and STS-MAC is that in the former the MAC algorithm is applied to the message that is signed, rather than to the signature of the message.

We note that Bellare, Canetti and Krawczyk [5] have recently provided a model and security definitions under which ISO-STS-MAC without the inclusion of the MACs is provably secure. How their model compares with the model of [9] is not entirely clear.

3.3 Other UKS attacks

The on-line UKS attacks of §3.2 cannot, in general, be launched on STS-ENC because the signatures $S_A(\alpha^{r_A}, \alpha^{r_B})$ and $S_B(\alpha^{r_B}, \alpha^{r_A})$ are not known by the adversary. Is it possible to extend the attacks to provide UKS attacks on STS-ENC that cannot be prevented by checking knowledge of private keys during certification? This section suggests a possible (although unlikely) scenario in which such (off-line) attacks on STS-ENC (and STS-MAC) may be successful. The attack illustrates two points:

1. A complete description of STS-ENC should include a complete specification of the underlying symmetric-key encryption and signature schemes, together with a statement of the security properties they are assumed to possess; and

2. Performing public-key validation [19] of signature keys is a sensible measure to take. (Rationale for performing key validation of public keys for use in Diffie-Hellman-based key agreement protocols is provided in [23].)

The attack is similar to the attack presented in §3.2, but relies on the following assumption on the signature scheme: E is able to certify a key pair (P_E, S_E) such that A's signature on any message M is also valid as E's signature on message M. Note that deterministic signature schemes cannot possess this property and be secure, since E knows S_E and can therefore compute

A's signatures using S_E. However it is possible that some probabilistic signature schemes possess this property. This is illustrated by the following example.

Suppose that the underlying signature scheme is the ElGamal signature scheme (see §4.3). Suppose that entities select their own domain parameters p and g, as may be the case in high security applications. Suppose further that when certifying an entity E's public key $P_E = (p, g, y)$ (where $y = g^e \pmod p$ and e is E's private key), the CA does not perform public-key validation; that is, the CA does not verify that p, g and y possess the requisite arithmetic properties — that p is prime, g is a generator of $\mathbb{Z}_p^*$, and $1 \leq y \leq p-1$. Finally, suppose that the CA verifies that E possesses the private key corresponding to its public key by asking E to sign a challenge message.

If a dishonest entity E selects $g = 0$ (which is *not* a generator of $\mathbb{Z}_p^*$), then $y = 0$. In this case, every pair of integers (r, s), where $1 \leq r \leq p-1$ and $1 \leq s \leq p-2$, is a valid signature for E on any message M since the ElGamal signature verification equation (see §4.3) $g^m \equiv y^r r^s \pmod p$ is satisfied. In particular, if the CA does not validate E's public key, then it will accept E's proof of possession of its private key.

Having obtained a certificate Cert_E of such an invalid public key $P_E = (p, 0, 0)$ (where the prime p is greater than the prime moduli of A and B), E can now launch UKS attacks against the responser or the initiator in both STS-ENC and STS-MAC in exactly the same way as described in §3.1. For example, in the attack against the initiator, E replaces A's identity with its own identity in flow (1), and then replaces Cert_A with Cert_E in flow (3). Note that these are not on-line attacks since E can get its public key certified in advance of the attack. Note also that these attacks are different from the public key substitution attacks of §3.1, because in the former E has indeed demonstrated possession of its private key to the CA during the certification process.

As precautionary measures, we recommend that public-key validation of signature keys be performed, and that STS-ENC be modified so that either the flow number and identities of the sender and intended recipient are included in the signed message[3] or that the identities be included in the key derivation function (as in item 5 in §3.2).

4 Duplicate-signature key selection

This section examines whether commonly used signature schemes possess the duplicate-signature key selection property that is required in §3.2: given A's public key P_A for a signature scheme S, and given A's signature s_A on a message M, can an adversary select a key pair (P_E, S_E) for S such that s_A is also E's signature on the message M? We demonstrate that, in certain circumstances, the RSA [31], Rabin [30], ElGamal [12], DSA [1, 27], and ECDSA [3] signature schemes all possess this property. In the RSA scheme, it is assumed that each entity is permitted to select its own encryption exponent e. In the ElGamal, DSA and ECDSA schemes, it is assumed that entities are permitted to select their own domain parameters; this is what might be done in high security applications.

It must be emphasized that possession of the duplicate-signature key selection property does

[3]The resulting revised protocols are the same as the ones presented in items 3 and 4 in §3.2 with the data $(S_A(m), \mathrm{MAC}_K(S_A(m)))$ replaced by $E_K(S_A(m))$, and $(S_B(m), \mathrm{MAC}_K(S_B(m)))$ replaced by $E_K(S_B(m))$.

not constitute a weakness of the signature scheme — the goal of a signature scheme is to be existentially unforgeable against an adaptive chosen-message attack [13].

In the following, H denotes a cryptographic hash function such as SHA-1 [28].

4.1 RSA

Key pair: A's public key is $P_A = (N, E)$, where N is a product of two distinct primes P and Q, and $1 < E < \Phi$, $\gcd(E, \Phi) = 1$, where $\Phi = (P-1)(Q-1)$. A's private key is D, where $1 < D < \Phi$ and $ED \equiv 1 \pmod{\Phi}$.

Signature generation: To sign a message M, A computes $m = H(M)$ and $s = m^D \bmod N$. A's signature on M is s. Here, H may also incorporate a message formatting procedure such as the ones specified in the ANSI X9.31 [2], FDH [8] and PSS [8] variants of RSA.

Signature verification: Given an authentic copy of A's public key, one can verify A's signature s on M by computing $m = H(M)$, and verifying that $s^E \equiv m \pmod{N}$.

Adversary's actions: Given A's public key P_A and A's signature s on M, E does the following.

1. Compute $m = H(M)$.
2. Select a prime p such that:
   (a) $p-1$ is smooth; and
   (b) s and m are both generators of $\mathbb{Z}_p^*$.
3. Select a prime q such that:
   (a) $pq > N$;
   (b) $q-1$ is smooth;
   (c) $\gcd(p-1, q-1) = 2$; and
   (d) s and m are both generators of $\mathbb{Z}_q^*$.
4. Since $p-1$ is smooth, E can use the Pohlig-Hellman algorithm [29] to efficiently find an integer x_1 such that $s^{x_1} \equiv m \pmod{p}$.
5. Similarly, since $q-1$ is smooth, E can efficiently find an integer x_2 such that $s^{x_2} \equiv m \pmod{q}$.
6. Compute $n = pq$, $\phi = (p-1)(q-1)$, and $\lambda = \phi/2$.
7. Find the unique integer e, $1 < e < \lambda$, such that $e \equiv x_1 \pmod{(p-1)}$ and $e \equiv x_2 \pmod{(q-1)}$. This can be done by first solving the congruence

$$t(p-1)/2 \equiv (x_2 - x_1)/2 \pmod{(q-1)/2}$$

   for t (note that $x_2 - x_1$ is indeed even), and then setting $e = x_1 + t(p-1) \bmod \lambda$. Note also that since m is a generator of $\mathbb{Z}_p^*$, we have $\gcd(x_1, p-1) = 1$; similarly $\gcd(x_2, q-1) = 1$. If follows that $\gcd(e, \phi) = 1$.

8. Compute an integer d, $1 < d < \phi$, such that $ed \equiv 1 \pmod{\phi}$.

9. E forms $P_E = (n, e)$; E's private key is d.

Observe that s is also E's signature on M since

$$s^e \equiv s^{e \bmod (p-1)} \equiv s^{x_1} \equiv m \pmod{p}$$

and

$$s^e \equiv s^{e \bmod (q-1)} \equiv s^{x_2} \equiv m \pmod{q},$$

whence

$$s^e \equiv m \pmod{n}.$$

Remarks

1. The following is a heuristic analysis of the expected number of candidate p's and q's that are chosen before primes satisfying the conditions in steps 2 and 3 are found.

   Suppose that the desired bitlength of both p and q is k. Candidates p and q can be selected by first choosing $p-1$ and $q-1$ to be products of small prime powers (thus ensuring conditions 2(a) and 3(b)); the primes occurring in the two products should be pairwise distinct, except for a 2 which occurs exactly once in each product (this ensures that $\gcd(p-1, q-1) = 2$). The candidate p is then subjected to a primality test. By the prime number theorem [25, Fact 2.95], the expected number of trials before a prime p is obtained is $(\frac{1}{2}\ln 2)k$. Given that p is prime, the probability that both m and s are generators of $\mathbb{Z}_p^*$ is (see [25, Fact 2.102])

   $$\left(\frac{\phi(p-1)}{(p-1)}\right)^2 < \left(\frac{1}{6\ln\ln(p-1)}\right)^2.$$

   If either m or s does not generate $\mathbb{Z}_p^*$, then another candidate p is selected. Hence, the expected number of trials before an appropriate p is found is

   $$\left(\frac{1}{2}\ln 2\right) k(6\ln\ln(p-1))^2 = O(k(\ln k)^2).$$

   It follows that the expected number of candidates p and q before appropriates primes are found is also $O(k(\ln k)^2)$.

2. Observe that (n, e) is a valid RSA public key, and that E knows the corresponding private key d.

3. To reduce the amount of on-line work required, the adversary could use A's public key to precompute several candidate pairs of primes p and q which satisfy conditions 2(a), 3(a), 3(b), and 3(c). Subsequently, when the adversary sees A's signature s on M, it can choose a precomputed pair of primes which also satisfy conditions 2(b) and 3(d).

4.2 Rabin

Key pair: A's public key is $P_A = N$, where N is a product of two distinct primes P and Q. A's private key is (P, Q).

Signature generation: To sign a message M, A computes $m = H(M)$, and finds a square root s of m modulo N: $s^2 \equiv m \bmod N$. A's signature on M is s. (If m is not a quadratic residue modulo N, then m should be adjusted in a predetermined way so that the result is one.)

Signature verification: Given an authentic copy of A's public key, one can verify A's signature s on M by computing $m = H(M)$, and verifying that $s^2 \equiv m \pmod N$.

Adversary's actions: Given A's public key P_A and A's signature s on M, E computes $n = (s^2 - m)/N$ and forms $P_E = n$. Observe that s is also E's signature on M since $s^2 \equiv m \pmod n$.

Remarks

1. The bitlength of n is expected to be the same as the bitlength of N.

2. n is most likely not the product of two distinct primes, and hence is not a valid Rabin public key. (Assuming that n is a random k-bit integer, the expected total number of prime factors of n is approximately $\ln k$; [25, Fact 3.7(iii)].) However, it is difficult, in general, to test whether a composite integer is a product is of two distinct primes; hence Rabin public-key validation is usually not performed in practice.

3. Assuming that n is a random k-bit integer, the probability that the bitlength of the second-largest prime factor of n is $\leq 0.22k$ is about $\frac{1}{2}$ [25, Fact 3.7(ii)]. Thus, for example, if 512-bit moduli are being used, then the probability that the bitlength of the second-largest prime factor of n is ≤ 113 is about $\frac{1}{2}$. Such n can be readily factored with the elliptic curve factoring algorithm [22]. Given the prime factorization of n, E can hope to convince the CA that it knows the corresponding private key (even though one may not exist — n may not be a product of 2 distinct primes), by signing (computing square roots modulo n, as with the Rabin scheme) a message of the CA's choice.

4.3 ElGamal

Domain parameters: A safe prime p (i.e., $q := (p-1)/2$ is prime), and a generator g of $\mathbb{Z}_p^*$.

Key pair: A's private key is an integer a, $1 \leq a \leq p-2$. A's public key is $P_A = (p, g, y)$, where $y = g^a \bmod p$.

Signature generation: To sign a message M, A selects a random integer k, $1 \leq k \leq p-2$, such that $\gcd(k, p-1) = 1$, and computes $m = H(M)$, $r = g^k \bmod p$, and $s = k^{-1}(m - ar) \bmod (p-1)$. A's signature on M is (r, s).

Signature verification: Given an authentic copy of A's public key, one can verify A's signature (r, s) on M by computing $m = H(M)$, and verifying that $g^m \equiv y^r r^s \pmod p$.

Adversary's actions: Given A's public key P_A and A's signature (r, s) on M, E does the following. If $\gcd(s, p-1) \neq 1$ or if $\gcd(m, r) = 2$ or q, then E terminates with failure. Otherwise,

E selects an arbitrary integer c, $1 \leq c \leq p-2$, such that $\gcd(t, p-1) = 1$, where $t = m - cr$. E then computes $\overline{g} = (r^s)^{t^{-1} \bmod (p-1)} \bmod p$ and forms $P_E = (p, \overline{g}, \overline{y})$, where $\overline{y} = \overline{g}^c \bmod p$.

Observe that (r, s) is also E's signature on M since

$$\overline{g}^{(-m)} \overline{y}^r r^s \equiv \overline{g}^{-(m-cr)} r^s \equiv (r^s)^{-t^{-1} t \bmod (p-1)} r^s \equiv 1 \pmod p.$$

Remarks.

The condition $\gcd(s, p-1) = 1$ ensures that r^s, and hence also $\overline{g}$, is a generator of $\mathbb{Z}_p^*$. The condition $\gcd(m, r) \neq 2, q$ ensures that there exists a c for which $\gcd(t, p-1) = 1$; it also implies that a non-negligible proportion of all c's satisfy $\gcd(t, p-1) = 1$. If we make the heuristic assumption that r, s and m are distributed uniformly at random from $[1, p-1]$, then we see that the success probability of the adversary is about $\frac{3}{8}$.

4.4 DSA

Domain parameters: Primes p and q such that q divides $p-1$, and an element $g \in \mathbb{Z}_p^*$ of order q. Typically p has bitlength 1024 and q has bitlength 160.

Key pair: A's private key is an integer a, $1 \leq a \leq q-1$. A's public key is $P_A = (p, q, g, y)$, where $y = g^a \bmod p$.

Signature generation: To sign a message M, A selects a random integer $k \in [1, q-1]$, and computes $m = H(M)$, $r = (g^k \bmod p) \bmod q$, and $s = k^{-1}(m + ar) \bmod q$. A's signature on M is (r, s).

Signature verification: Given an authentic copy of A's public key, one can verify A's signature (r, s) on M by computing $m = H(M)$, $u_1 = s^{-1}m \bmod q$, $u_2 = s^{-1}r \bmod q$, and verifying that $r = (g^{u_1} y^{u_2} \bmod p) \bmod q$.

Adversary's actions: Given A's public key P_A and A's signature (r, s) on M, E selects a random integer $c \in [1, q-1]$ such that $t := ((u_1 + cu_2) \bmod q) \neq 0$. E then computes $r_1 = g^{u_1} y^{u_2} \bmod p$ and $\overline{g} = r_1^{t^{-1} \bmod q} \bmod p$, and forms $P_E = (p, q, \overline{g}, \overline{y})$ where $\overline{y} = \overline{g}^c \bmod p$. Note that $\mathrm{ord}(\overline{g}) = q$, so P_E is a valid DSA public key.

Observe that (r, s) is also E's signature on M since

$$\overline{g}^{u_1} \overline{y}^{u_2} \equiv \overline{g}^{u_1 + cu_2} \equiv \overline{g}^t \equiv r_1 \pmod p,$$

whence $r = (\overline{g}^{u_1} \overline{y}^{u_2} \bmod p) \bmod q$.

4.5 ECDSA

ECDSA is the elliptic curve analogue of the DSA and is specified in [3].

Domain parameters: An elliptic curve E defined over the finite field $\mathbb{F}_q$ with $\#E(\mathbb{F}_q) = nh$ and n prime, and a point $P \in E(\mathbb{F}_q)$ of order n.

Key pair: A's private key is an integer a, $1 \leq a \leq n-1$. A's public key is $P_A = (p, E, n, P, Q)$, where $Q = aP$.

Signature generation: To sign a message M, A selects a random integer k, $1 \le k \le n-1$, and computes $m = H(M)$, $R = kP$, $r = x(R) \bmod n$, and $s = k^{-1}(m + ar) \bmod n$. Here, $x(R)$ denotes the x-coordinate of the point R. A's signature on M is (r, s).

Signature verification: Given an authentic copy of A's public key, one can verify A's signature (r, s) on M by computing $m = H(M)$, $R = s^{-1}mP + s^{-1}rQ$, and verifying that $r = x(R) \bmod n$.

Adversary's actions: Given A's public key P_A and A's signature (r, s) on M, E selects an arbitrary integer c, $1 \le c \le n-1$, such that $t := ((s^{-1}m + s^{-1}rc) \bmod n) \ne 0$. E then computes $R = s^{-1}mP + s^{-1}rQ$ and $\overline{P} = (t^{-1} \bmod n)R$, and forms $P_E = (p, E, n, \overline{P}, \overline{Q})$, where $\overline{Q} = c\overline{P}$. Note that $\mathrm{ord}(\overline{P}) = n$, so P_E is a valid ECDSA public key.

Observe that (r, s) is also E's signature on M since

$$s^{-1}m\overline{P} + s^{-1}r\overline{Q} = (s^{-1}m + s^{-1}rc)\overline{P} = t\overline{P} = R, \tag{1}$$

whence $r = x(R) \bmod n$.

Remarks.

E's domain parameters are the same as A's, with the exception of the base point $\overline{P}$. If the elliptic curve was chosen verifiably at random using a canonical seeded hash function (e.g., as specified in ANSI X9.62 [3]), then E can use the same (non-secret) seed as selected by A to demonstrate to the CA that the curve was indeed selected verifiably at random. There is no requirement in ANSI X9.62 for generating the base point verifiably at random. Hence, performing domain parameter validation as specified in ANSI X9.62 does not foil the adversary.

5 Conclusions

This paper presented some new unknown key-share attacks on the STS-MAC key agreement protocol. The attacks are a concern in practice since STS-MAC purports to provide both implicit key authentication and key confirmation. There are various ways in which the attacks can be circumvented. Our preferred way is to include flow numbers and identities in the messages being signed, and to separate keys used to provide key confirmation from derived shared secret keys.

Acknowledgements

The authors would like to thank Don Johnson and Minghua Qu for their valuable comments on earlier drafts of this paper.

References

[1] ANSI X9.30 (Part 1), *Public Key Cryptography Using Irreversible Algorithms for the Financial Services Industry – Part 1: The Digital Signature Algorithm (DSA)*, 1995.

[2] ANSI X9.31, *Digital Signatures Using Reversible Public Key Cryptography for the Financial Services Industry (rDSA)*, working draft, March 1998.

[3] ANSI X9.62, *The Elliptic Curve Digital Signature Algorithm (ECDSA)*, working draft, August 1998.

[4] ANSI X9.63, *Elliptic Curve Key Agreement and Key Transport Protocols*, working draft, October 1998.

[5] M. Bellare, R. Canetti and H. Krawczyk, "A modular approach to the design and analysis of authentication and key exchange protocols", *Proceedings of the 30th Annual Symposium on the Theory of Computing*, 1998. A full version of this paper is available at `http://www-cse.ucsd.edu/users/mihir`

[6] M. Bellare and P. Rogaway, "Random oracles are practical: a paradigm for designing efficient protocols", *1st ACM Conference on Computer and Communications Security*, 1993, 62–73. A full version of this paper is available at `http://www-cse.ucsd.edu/users/mihir`

[7] M. Bellare and P. Rogaway, "Entity authentication and key distribution", *Advances in Cryptology – Crypto '93*, LNCS **773**, 1993, 232-249. A full version of this paper is available at `http://www-cse.ucsd.edu/users/mihir`

[8] M. Bellare and P. Rogaway, "The exact security of digital signatures—how to sign with RSA and Rabin", *Advances in Cryptology – Eurocrypt '96*, LNCS **1070**, 1996, 399-416.

[9] S. Blake-Wilson, D. Johnson and A. Menezes, "Key agreement protocols and their security analysis", *Proceedings of the sixth IMA International Conference on Cryptography and Coding*, LNCS **1355**, 1997, 30-45. A full version of this paper is available at `http://www.cacr.math.uwaterloo.ca/`

[10] S. Blake-Wilson and A. Menezes, "Authenticated Diffie-Hellman key agreement protocols", *Proceedings of SAC '98*, LNCS, to appear.

[11] W. Diffie, P. van Oorschot and M. Wiener, "Authentication and authenticated key exchanges", *Designs, Codes and Cryptography*, **2** (1992), 107-125.

[12] T. ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithms", *IEEE Transactions on Information Theory*, **31** (1985), 469-472.

[13] S. Goldwasser, S. Micali, and R. Rivest, "A digital signature scheme secure against adaptive chosen message attacks", *SIAM Journal on Computing*, **17** (1988), 281-308.

[14] IPSEC Working Group, *The OAKLEY Key Determination Protocol*, Internet Draft, Internet Engineering Task Force, available from `http://www.ietf.cnri.reston.va.us/`

[15] ISO/IEC 8824-1, *Information Technology – Open Systems Interconnection – Abstract Syntax Notation One (ANS.1) – Part 1: Specification of Basic Notation.*

[16] ISO/IEC 8825-3, *Information Technology – Open Systems Interconnection – Specification of ASN.1 Encoding Rules – Part 3: Distinguished Canonical Encoding Rules.*

[17] ISO/IEC 9798-3, *Information Technology – Security Techniques – Entity Authentication Mechanisms – Part 3: Entity Authentication Using a Public-Key Algorithm* 1993.

[18] ISO/IEC 11770-3, *Information Technology – Security Techniques – Key Management – Part 3: Mechanisms Using Asymmetric Techniques*, draft, (DIS), 1996.

[19] D. Johnson, Contribution to ANSI X9F1 working group, 1997.

[20] B. Kaliski, Contribution to ANSI X9F1 and IEEE P1363 working groups, June 17 1998.

[21] L. Law, A. Menezes, M. Qu, J. Solinas, S. Vanstone, "An efficient protocol for authenticated key agreement", Technical report CORR 98-05, Department of C&O, University of Waterloo, 1998. Also available at `http://www.cacr.math.uwaterloo.ca/`

[22] H.W. Lenstra, "Factoring integers with elliptic curves", *Annals of Mathematics*, **126** (1987), 649-673.

[23] C. Lim and P. Lee, "A key recovery attack on discrete log-based schemes using a prime order subgroup", *Advances in Cryptology – Crypto '97*, LNCS **1294**, 1997, 249-263.

[24] A. Menezes, M. Qu and S. Vanstone, "Some new key agreement protocols providing mutual implicit authentication", *Workshop on Selected Areas in Cryptography (SAC '95)*, 22-32, 1995.

[25] A. Menezes, P. van Oorschot and S. Vanstone, *Handbook of Applied Cryptography*, CRC Press, 1997.

[26] C. Mitchell and A. Thomas, "Standardising authentication protocols based on public key techniques", *Journal of Computer Security*, **2** (1993), 23-36.

[27] National Institute of Standards and Technology, *Digital Signature Standard*, FIPS Publication 186, 1994.

[28] National Institute of Standards and Technology, *Secure Hash Standard (SHS)*, FIPS Publication 180-1, 1995.

[29] S. Pohlig and M. Hellman, "An improved algorithm for computing logarithms over $GF(p)$ and its cryptographic significance", *IEEE Transactions on Information Theory*, **24** (1978), 106-110.

[30] M.O. Rabin, "Digitalized signatures and public-key functions as intractable as factorization", MIT/LCS/TR-212, MIT Laboratory for Computer Science, 1979.

[31] R.L. Rivest, A. Shamir and L.M. Adleman, "A method for obtaining digital signatures and public-key cryptosystems", *Communications of the ACM*, **21** (1978), 120-126.

[32] P. van Oorschot, "Extending cryptographic logics of belief to key agreement protocols", *1st ACM Conference on Computer and Communications Security*, ACM Press, 1993, 232-243.