Okamoto-Tanaka Revisited: Fully Authenticated Diffie-Hellman with Minimal Overhead

Rosario Gennaro Hugo Krawczyk Tal Rabin

IBM T.J. Watson Research Center
Hawthorne, New York 10532*

Abstract. The Diffie-Hellman protocol (DHP) is one of the most studied protocols in cryptography. Much work has been dedicated to armor the original protocol against *active attacks* while incurring a minimal performance overhead relative to the basic (unauthenticated) DHP. This line of work has resulted in some remarkable protocols, e.g., MQV, where the protocol's communication cost is identical to that of the basic DHP and the computation overhead is small. Unfortunately, MQV and similar 2-message "implicitly authenticated" protocols do not achieve full security against active attacks since they cannot provide forward secrecy (PFS), a major security goal of DHP, against active attackers.

In this paper we investigate the question of whether one can push the limits of authenticated DHPs even further, namely, to achieve communication complexity as in the original DHP (two messages with a single group element per message), maintain low computational overhead, and yet achieve full PFS against active attackers in a *provable* way. We answer this question in the affirmative by resorting to an old and elegant key agreement protocol: the Okamoto-Tanaka protocol [32]. We present a variant of the protocol (denoted mOT) which achieves the above minimal communication, incurs a computational overhead relative to the basic DHP that is practically negligible, and yet achieves full provable key agreement security, including PFS, against active attackers. Moreover, due to the identity-based properties of mOT, even the sending of certificates (typical for authenticated DHPs) can be avoided in the protocol.

As additional contributions, we apply our analysis to prove the security of a recent multi-domain extension of the Okamoto-Tanaka protocol by Schridde et al. and show how to adapt mOT to the (non id-based) certificate-based setting.

* Email: rosario@us.ibm.com, hugo@ee.technion.ac.il, talr@us.ibm.com.

1 Introduction

Since the invention of the Diffie-Hellman protocol (DHP) [13], much work has been dedicated to armor the protocol against active ("man in the middle") attacks. Designing *authenticated Diffie-Hellman protocols* has proved to be very challenging at the design and analysis level, especially when trying to optimize performance (both computation and communication). This line of work has been important not only from the practical point of view but also for understandings what are the *essential limits* for providing authentication to the DHP.

In particular, it has been shown that one can obtain an *authenticated* DH protocol with the same communication as the basic unauthenticated DHP (at least if one ignores the transmission of public key certificates); namely, a 2-message exchange where each party sends a single DH value, and where the two messages can be sent in any order. A prominent example of such protocols is MQV [26] (and its provably-secure variant HMQV [25]) where the cost of computing a session key is as in the basic unauthenticated DHP plus half the cost of one exponentiation (i.e., one off-line exponentiation and 1.5 on-line exponentiations).

Protocols such as the 2-message MQV are "implicitly-authenticated protocols;" that is, the information transmitted between the parties is computed without access to the parties' long-term secrets while the authentication is accomplished via the computation of the session key that involves the long-term private/public keys of the parties. Unfortunately, implicitly-authenticated protocols, while offering superb performance, are inherently limited in their security against active attackers. Indeed, as shown in [25], such protocols can achieve perfect forward secrecy (PFS) *against passive attackers only.* Recall that PFS ensures that once a session key derived from a Diffie-Hellman value is erased from memory, there is no way to recover the session key even by an attacker that gains access to the long-term authentication keys of the parties after the session is established. PFS is a major security feature that sets DHPs apart from other key agreement protocols (such as those based in PK encryption) and is the main reason for the extensive use of DHPs in practice (e.g., IPsec and SSH). Adding PFS against active attackers to protocols like MQV requires increased communication in the form of additional messages and/or explicit signatures.

In this paper we investigate the theoretical and practical question of whether the limits of DHPs can be pushed further and obtain a protocol with full security against active attackers (including PFS) while preserving the communication complexity of a basic DHP (two messages with a single group element per message) and low computational overhead. We answer this question in the affirmative by departing from implicitly authenticated protocols and resorting to an old and elegant key agreement protocol: the Okamoto-Tanaka protocol [32]. We present a variant of the protocol (denoted mOT) which achieves the above minimal communication, incurs a negligible computational overhead relative to a basic DHP over an RSA group, and yet achieves *provable* security including *full PFS* against active attackers[1]. Moreover, due to the identity-based properties of mOT, even the sending and verification of certificates is avoided in the protocol.

1.1 Our Results

An analysis of the original Okamoto-Tanaka protocol shows that it is vulnerable to some forms of attacks, in particular known-key and malleability attacks. Yet, after introducing some simple but crucial hashing operations, we obtain a protocol, mOT, for which we present a rigorous proof of security in the Canetti-Krawczyk (CK) Key-Agreement Protocol model [6]. The security of the

[1] There are DH protocols that provide full PFS against active attacks with just two messages, but they require to send (and process) additional information, e.g. explicit signatures [40] or encrypted challenges [23].

protocol in this model, including weak PFS (i.e., against passive attacks only), can be proven in the random oracle model under *the standard RSA assumption.* For the proof of *full* PFS against active attackers (and only for this proof) we resort to non-black-box assumptions in the form of the "knowledge of exponent" assumptions. We provide more details now.

Modified Okamoto-Tanaka (mOT). Our modified Okamoto-Tanaka protocol, which we denote by mOT, is described in Figure 1 (for a precise specification see Section 3). We describe the protocol as an identity-based protocol using a KGC (key generation center) as this setting provides added performance advantages to the protocol. A certificate-based variant is presented in Section 6. The differences between mOT and the original Okamoto-Tanaka protocol include the hashing operation on identities as well as the hashing and squaring operations in the computation of the session key K. We show that omitting any one of the two hash operations renders the protocol insecure. As for the squaring operation, we use it in our proofs but we do not know of a specific attack if omitted.

The Modified Okamoto-Tanaka (mOT) Protocol

Setting: A Key Generation Center (KGC) chooses RSA parameters $N = pq$ (such that p and q are random safe primes), and exponents d, e, and a random generator g of QR_N, the (cyclic) subgroup of quadratic residues mod$N$.

KGC publishes N, e, g, two hash functions H (with range QR_N) and H' (with range of the desired length of the session key), and distributes to each user U with identity id_U a private key $S_U = H(id_U)^d \bmod N$.

Key agreement: A and B choose ephemeral private exponents x and y, respectively.

$A \xrightarrow{\quad \alpha = g^x S_A \bmod N \quad} B$

$A \xleftarrow{\quad \beta = g^y S_B \bmod N \quad} B$

$\bar{K}_A = (\beta^e / H(id_B))^{2x} \bmod N \qquad \bar{K}_B = (\alpha^e / H(id_A))^{2y} \bmod N$

$\bar{K} = \bar{K}_A = \bar{K}_B = g^{2xye} \bmod N$

A and B set the session key to $K = H'(\bar{K}, id_A, id_B, \alpha, \beta)$

Fig. 1. A and B share session key K. See Section 3 for full details.

Security Proof and Full PFS "for free". We present a rigorous proof of security of the mOT protocol in the Canetti-Krawczyk (CK) model of authenticated key agreement security specialized to two-message protocols as in [25]. We consider such a proof to be an essential part of the design of any protocol, especially given the long history of flawed key agreement protocols, particularly those that optimize computation and communication. We prove the basic properties of security in the CK model under the sole RSA assumption (namely, the hardness of inverting RSA on random inputs) in the random oracle model.[2] Also, under the same assumptions, we prove additional security properties in the extended CK model from [25], namely, resilience to KCI attacks[3] and weak PFS.

[2] As it can be readily seen, in the mOT protocol the ephemeral Diffie-Hellman values (i.e., any of x, y, g^x, g^y in Figure 1) must be guarded as well as the private keys of the parties (this is similar to the requirement in DSA signatures to prevent the disclosure of the ephemeral randomness value k); therefore, the analysis is carried under the assumption that g^x, g^y and long-term private keys are equally protected.

[3] Resistance to KCI means that even with the knowledge of Alice's private key the attacker cannot impersonate an honest party to Alice (though it can, of course, impersonate Alice to other parties).

The security result that sets our protocol and work apart, however, is our proof of *full PFS* for mOT, namely, *perfect forward secrecy against fully active attackers.* The proof of full PFS (and only this proof) requires two additional "non-black-box" assumptions: one is the well-known KEA1 (knowledge of exponent) assumption [11, 2] related to the hardness of the Diffie-Hellman problem and the second is similar in spirit but applies to the discrete logarithm problem (see Section 4). Enjoying full PFS is a major advantage of mOT relative to efficient two-message protocols such as MQV that can only offer weak PFS. Indeed, in spite of mOT transmitting a single group element in each of the two messages, it overcomes the inherent PFS limitations of implcitly authenticated DHPs by involving the sender's private key in the computation of each protocol's message. Most importantly, as we explain below, this *full security against active attackers* is achieved with zero communication and negligible computational overhead relative to the basic DHP. We believe this to be not just a practical feature of mOT but also a significant contribution to the theory of key agreement protocols showing that armoring the original DHP against active attackers can be achieved essentially "for free".

Performance. The cost of mOT remains essentially the same as in the basic (unauthenticated) DHP: one message per party, that can be sent in any order, with each message containing a single group element. No additional authentication information needs to be transmitted. Thanks to the identity-based properties of the protocol, public-key certificates need not be sent or verified. The only extra operation is one exponentiation to the e-th power, which can be chosen to be 3, and one squaring; that is, just three modular multiplications in all. However, note that mOT works over an RSA group and therefore exponentiations are more expensive than over elliptic curves (where protocols like MQV can be run). Yet, we also note that mOT can be implemented with short exponents, say 160-bit exponents when the modulus is of size 1024 (or a 224-bit exponent with a 2048-bit modulus). Our proof of the protocol holds in this case under the common assumption that in the RSA group the discrete logarithm problem remains hard also for these exponent sizes (see Section 3.4). In terms of practical efficiency, for moderate security parameters (160-200 bit exponents) the cost of one on-line exponetiation in mOT is competitive with the 1.5 exponentiations over elliptic curves required by MQV. For larger security parameters the advantage is fully on the elliptic curve side though in this case one has to also consider the overhead incurred by certificate processing in a protocol like MQV (which is costly especially for ECDSA-signed certificates).

Of course, beyond the practical performance considerations, mOT holds a significant security advantage over 2-message MQV, namely, its full PFS against active attackers. The fact that mOT can do so well with almost no overhead over the underlying basic Diffie-Hellman protocol, and with full security against active attacks, is an important theoretical (and conceptual) aspect of our work pointing to the limits of what is possible in this area.

The Need for a Key Generation Center (KGC). As an identity-based protocol, mOT avoids the need for certificates (a significant communication and computational advantage). The id-based setting, however, introduces the need for a KGC that generates and distributes keys to users. This results in a different trust model than the traditional certification authority (CA) that certifies public keys but does not generate or know the private keys of parties. Note, however, that in mOT the private keys are used only for authentication. Thus, while a KGC can impersonate a party, it cannot learn keys exchanged by that party (we note that the PFS property holds also against a corrupted KGC!). Note that a regular CA can also impersonate parties at will by issuing certificates with the user's name but with a private key known to the CA. Interestingly, as we show below, mOT can be modified to work also in the traditional CA setting.

Further contributions (multi-domain and certificate-based extensions). In Section 5 we extend the above proofs and analysis to a recent extension of the Okamoto-Tanaka protocol proposed by Schridde, Smith and Freisleben [37] that allows the execution of the protocol between users that belong to different domains, i.e., to different key generation centers (KGC). A user willing to communicate with a peer from another domain needs to obtain the peer's domain public parameters but it requires no per-user certificates. In the scheme from [37] (applied to our modified Okamoto-Tanaka protocol), each KGC chooses its own public parameters exactly as in the mOT protocol *independently* of other KGCs. There is *no* need for coordination between different KGCs (this is better than other inter-domain schemes in the literature that require the KGCs to share some common parameters such as a single elliptic curve and generator point). Importantly, in this extension the protocol between users of the same domain is exactly as in the basic mOT protocol with the same good performance. When users from two domains interact the protocol is performed similarly but requires twice as many operations. Multi-domain protocols have important real-world applications, including multi-domain Internet applications [37, 41] and applications to tactical and ad-hoc networks.

Finally, in Section 6 we show that the mOT protocol and its extension from [37] can be modified to work as "traditional" (i.e., not ID-based) key agreement protocol. We have two proposals: the first uses the mOT protocol (and inherits its outstanding efficiency) with a trusted key generation center to generate the composite N for all the parties. However, in contrast to the ID-based case, this center can disappear (in particular, erase all the trapdoor information) immediately after publishing N; users choose their own (random) private/public keys based on the public parameters only. The second proposal uses the protocol from [37] where each party chooses its own modulus N (leveraging on the multi-domain feature of this protocol) and hence completely dispenses of the initial trusted key generation center. The resultant protocol transmits twice as many bits as the id-based mOT but requires only 25% increase in computation.

Related work. Key agreement protocols (KAPs) have played an important role in the development of identity-based cryptography, with Okamoto [31], Okamoto and Tanaka [32], Gunther [16] being early examples of id-based cryptography. (Even earlier, the work by Blom [3] on key distribution can be seen as a precursor of id-based schemes.) With the flourishing of pairings-based cryptography, many more id-based KAPs have been designed; yet getting them right has been a challenging task. See the survey by Boyd and Choo [4] and Chen, Cheng, and Smart [8] for good descriptions and accounts of the main properties of many of these protocols. Even to date it seems that very few (e.g., [5, 42]) were given full proofs of security (many others were broken or enjoy only a restricted notion of security, such as partial resistance to known-key attacks). Also in the case of the *original* Okamoto-Tanaka protocol one can find several papers [31, 32, 27, 22] with partial or flawed analysis of the protocol[4]. In all, the mOT protocol studied here compares very favorably with other id-based and traditional KAPs in provability and security properties (e.g., PFS) as well as performance-wise.

Multi-domain extensions of id-based KAPs have been proposed in [9, 29] but without full proofs of security. The multi-domain extension of the mOT protocol that we fully analyze here is from Schridde et al. [37] which also contains a good discussion of the benefits of multi-domain identity-based protocols. In general, while interactive authenticated KAPs, especially those authenticated with signatures, can easily accommodate certificates (which a party can send together with its signature), avoiding the need for certificates constitutes a significant practical simplification of many systems. In particular, they provide more convenient solutions for revocation and less management

[4] Indeed, given the actual vulnerabilities of the protocol that we uncovered in our analysis of the original Okamoto-Tanaka protocol, these analyses cannot prove the security of the protocol except in very limited security models.

burden [41]. The Okamoto-Tanaka example shows that the identity-based setting can sometimes even improve performance.

Open questions. We believe that the mOT protocol is remarkable for its "minimalism", providing full and provable authenticated key-agreement security (including full PFS) with the same communication and minimal computational overhead relative to the underlying unauthenticated DHP. Yet there are several ways one could hope to improve on this protocol and on our results; achieving any of these improvements would bring us even closer to the "ultimate" authenticated DHP:
(i) Find a protocol with the same communication/computation/security characteristics as mOT but which works over arbitrary dlog groups (in particular elliptic curves). In this case, the minimalism of mOT would translate into optimal practical performance (even a certificate-based protocol with these properties would be very useful). (ii) Prove the full PFS security of mOT without resorting to non-black-box assumptions (while we believe that proofs under these assumptions carry a very strong evidence of security, using more standard assumptions is obviously desirable). (iii) Improve on mOT by avoiding the vulnerability of the protocol to the exposure of the DH values g^x, g^y.

Organization. After presenting some technical preliminaries in Section 2, we specify and analyze the mOT protocol in Section 3. This analysis is based on the Canetti-Krawczyk model of key-agreement security that is succinctly recalled in Appendix A. Section 4 presents the proof of full PFS for mOT and the assumptions used in that proof. In Section 5 we present a multi-authority extension of mOT (based on the work of [37]) and our analysis of the scheme. Finally, in Section 6 we show how to adapt mOT to the traditional certificate-based authentication model.

2 Preliminaries

Let $SPRIMES(n)$ be the set of n-bit long safe primes. Recall that a prime p is safe if $\frac{p-1}{2}$ is prime. Let $N = pq$ be the product of two random primes in $SPRIMES(n)$; denote $p = 2p' + 1$ and $q = 2q' + 1$. Let e be an integer which is relatively prime to $\phi(N) = 4p'q'$.

We say that the RSA Assumption (with exponent e) holds if for any probabilistic polynomial time adversary $\mathcal{A}$ the probability that $\mathcal{A}$ on input N, e, R, where $R \in_R Z_N^*$, outputs x such that $x^e = R \bmod N$ is negligible in n. The probability of success of $\mathcal{A}$ is taken over the random choices of p, q, R and the coin tosses of $\mathcal{A}$.

Remark (semi-safe primes). For simplicitly, we assume that $p = 2p' + 1$ and $q = 2q' + 1$ are safe primes, namely, p' and q' are prime. We can relax this assumption to require that $\gcd(p', q') = 1$ and that neither p' or q' have a prime factor smaller than 2^ℓ for a given security parameter ℓ. With these assumptions we get that QR_N is cyclic and that a random element in QR_N is a generator with overwhelming $(1 - 2^{-\ell})$ probability, two properties that we use in our construction and analysis.

Througout the paper we use the following well-known result of Shamir [35]:

Lemma 1. *Let N, e, d be RSA parameters and f be an integer relatively prime to e. There is an efficient procedure that given N, e, f (but not d) and a value $(x^f)^d \bmod N$, for $x \in Z_N^*$, computes $x^d \bmod N$.*

Proof. Since $gcd(e, f) = 1$ then one can compute integers a, b such that $ae + bf = 1$. Let $s = (x^f)^d$, then $\bar{s} = x^a s^b \bmod N$ satisfies $\bar{s}^e = x^{ae}(s^e)^b = x^{ae}(x^f)^b = x^{ae+bf} = x$, hence $x^d = \bar{s} \bmod N$. □

The cyclic group QR_N. If N is an RSA modulus product of safe primes, then the subgroup QR_N of quadratic residues in Z_N^* is cyclic of order $p'q'$. Let g be a random generator of QR_N (such

generator can be found by squaring a random element in Z_N^* (this algorithm yields a generator with overwhelming probability and the resulting distribution is statistically close to uniform). In protocols and proofs below we are going to generate random elements in the group generated by g according to the uniform distribution and with known exponents (i.e., their dlog to the base g). Such a random element X could be generated by choosing an integer $x \in [1..p'q']$ uniformly at random and setting $X = g^x \bmod N$. But this option implies knowledge of the factorization of N (knowing the value $p'q'$ is equivalent to factoring N). Parties who do not know the value $p'q'$ can approximate the uniform distribution over $\langle g \rangle$ as follows: generate $x \in [1..\lfloor N/4 \rfloor]$ and set $X = g^x \bmod N$. It is not hard to see (cf. [12]) that if p' and q' are of the same size (as required here) then the uniform distributions over $[1..p'q']$ and $[1..\lfloor N/4 \rfloor]$ are statistically close with an exponentially small gap.

Let g be a random generator of QR_N and let $X = g^x \bmod N$ and $Y = g^y \bmod N$ two random elements in QR_N. We say that the Computational Diffie-Hellman (CDH) Assumption (for N and g) holds if for any probabilistic polynomial time adversary $\mathcal{A}$ the probability that $\mathcal{A}$ on input N, g, X, Y outputs Z such that $Z = g^{xy} \bmod N$ is negligible in n. The probability of success of $\mathcal{A}$ is taken over the random choices of p, q, x, y and the coin tosses of $\mathcal{A}$. We know from [38] that the hardness of factoring N (and therefore the RSA Assumption) implies the CDH Assumption.

3 The Modified Okamoto-Tanaka Protocol

In this section we describe in full detail the modified version of the Okamoto-Tanaka protocol and provide rationale for the modifications made to the original protocol from [32]. Throughout the paper we refer to this protocol as the "Modified Okamoto-Tanaka Protocol (mOT)".

Protocol Setup. A key generation center KGC (for "trusted authority") chooses an RSA key (N, e, d), where N is the product of two safe primes p, q. As usual e, d are such that $ed = 1 \bmod \phi(N)$. The KGC also chooses a random generator g for the subgroup of quadratic residues QR_N. The public key of the KGC is (N, e, g) and its secret key is d.

Two hash functions H, H' are public parameter. The first function H outputs elements in the group generated by g (this can be achieved by setting H to be the square mod$N$ of another hash function with range Z_N^*). The second hash function H' outputs k-bit strings, where k is the length of the required session key.

Each user in the system has an identity; for convenience we sometimes associate a name to an identity. For example, Alice will be the name of a party while her identity is denoted id_A. We also denote $A = H(id_A)$ and $B = H(id_B)$ (thus, A, B are elements in QR_N). When Alice requests her secret key from the KGC, she receives the value $S_A = A^d \bmod N$ as her secret key (we can think of this as the KGC's RSA signature on Alice's id).

The key agreement. Alice chooses a random integer x from a set S that we specify below. She then computes $X = g^x \bmod N$ and sends to Bob $\alpha = X \cdot S_A \bmod N$. Bob chooses a random y in S, sets $Y = g^y \bmod N$, and sends to Alice the value $\beta = Y \cdot S_B \bmod N$. Alice computes a shared secret value $\bar{K}_A = (\beta^e / B)^{2x} \bmod N$ while Bob computes it as $\bar{K}_B = (\alpha^e / A)^{2y} \bmod N$. (Alice and Bob check that the incoming value is in Z_N^* or else they abort the session.)

Notice that the values $\bar{K}_A, \bar{K}_B$ are equal:

$$\bar{K}_A = Y^{2xe} \cdot (S_B^e / B)^{2x} = Y^{2xe} (B^{ed} / B)^{2x} = g^{2xye} = ((XS_A)^e / A)^{2y} = \bar{K}_B$$

Alice and Bob set $\bar{K}$ as this shared secret value $\bar{K} = \bar{K}_A = \bar{K}_B = g^{2xye} \bmod N$ and set the session key to $K = H'[\bar{K}, id_A, id_B, \alpha, \beta]$. Since we want both parties to compute the same session key we

need to determine an ordering between id_A and id_B, and between α and β in the input to the hash; for example, a lexicographic ordering (note that we are not assuming necessarily that there is a definite role of "initiator" and "responder" in the protocol, and hence we do not use such roles to determine the ordering of the above values).

Protection of ephemeral values. We specify that the ephemeral Diffie-Hellman values X, Y chosen by the parties be given the same protection level as the private keys S_A, S_B (indeed, learning these ephemeral values is equivalent to learning the private keys). In particular, if these values are stored in the (less secure) session state they need to be stored encrypted under a (possibly symmetric) key stored with the private key. (This is analogous to the need for protecting the ephemeral value k in a DSA signature.) In addition, we specify that in the computation of the session key, the hashing of the value $\bar{K}$ be performed in protected memory and only the session key be exported to a session state or application (learning the shared secret $\bar{K}$ value opens some attack venues as explained in Section 3.1).

The exponents set S and performance considerations. The performance cost of the protocol is dominated by the exponentiation operations; hence the choice of the set S from which ephemeral exponents are selected is important. A first choice would be to define S as the interval $[1..\lfloor\sqrt{N}/2\rfloor]$. In this case, as shown in [15] (following the results in [18]), the two distributions

$$\{g^x \text{ for } x \in_R [1..\lfloor N/4\rfloor]\} \quad \text{and} \quad \{g^x \text{ for } x \in_R [1..\lfloor\sqrt{N}/2\rfloor]\}$$

are computationally indistinguishable under the assumption that factoring N is hard. Therefore, one can use exponents of length *half the modulus* without any loss in security. However, performance can be significantly improved by setting S to be the set of exponents of length κ, where κ is twice the security parameter (e.g., $\kappa = 224$). Indeed in this case, the security of the protocol relies on the common assumption that discrete log (over Z_N^*) is hard also when the exponents are of length κ. Indeed (see Lemma 3.6 in [15]), this assumption implies that the two distributions

$$\{g^x \text{ for } x \in_R [1..\lfloor N/4\rfloor]\} \quad \text{and} \quad \{g^x \text{ for } x \in_R [1..2^\kappa]\}$$

are computationally indistinguishable, and hence using the short or long exponents is equivalent. Therefore, we recommend the protocol to be implemented using short exponents; in particular, we use this case when discussing the protocol's performance.

3.1 Rationale for the Modifications of the Original mOT Protocol

In our description of the mOT protocol we introduced a few modifications of the original Okamoto-Tanaka protocol, namely, the hashing of the identity prior to computing the private keys (i.e., the hashing in the computation of $S_A = H(id_A)^d \bmod N$) and the hashing operation $H'(\bar{K}, id_A, id_B, \alpha, \beta)$ in the definition of the session key. While the need for the former hash is quite clear (it is somewhat hinted to in [22] and included in the variant of the protocol in [37]), the need for the second hashing is more subtle. And in fact, the second hash does not appear in [32] and follow-up work (including [22] and other works that claimed the protocol's security); they defined the session key to be the value $(\beta^e/B)^x \bmod N = (\alpha^e/A)^x \bmod N$ without the hashing operation. As we see next, both hashing steps are essential for the security of the protocol and, in particular, for proving its security. Specifically, we show how the omission of any of these hashing operations renders the protocol vulnerable to attacks.

The attacks in items 1 and 2 below show the necessity of hashing the ids of the parties while item 3 and 4 demonstrates an attack based the lack of the hash on the derived key value (as well as the session identifier).

1. If the id is not hashed before the computation of the user's secret key, an attacker can learn Alice's private key. The attacker sees Alice's id id_A and sets its own id as party P to $id_P = id_A r^e$ for some random r. It requests its secret key form the KGC and receives $S_P = (id_A r^e)^d \bmod N = r \cdot S_A \bmod N$, and therefore can learn the value $S_A = id_A^d$ by dividing r out.

2. If ids are not hashed then an Unknown Key Share (UKS) attack [14] is possible. The attacker sees Alice's id id_A and sets an id for (adversarial) party P to the random value $id_P = id_A g^{te} \bmod N$ where g and e are the systems parameters and t a random exponent in $[1..\lfloor N/4 \rfloor]$. When Alice sends $(id_A, \alpha = XS_A)$ to Bob, P replaces it with (id_P, π) where $\pi = \alpha g^t = XS_A g^t$ which is a "correct" message coming from P on the ephemeral value X (indeed P's secret key is $S_P = S_A g^t$). When Bob responds with $(id_B, \beta = YS_B)$ this value is simply forwarded unchanged by P to Alice. Now Alice computes $\bar{K} = (\beta^e / id_B)^{2x} = ((YS_B)^e / id_B)^{2x} = (Y^e id_B^{ed} / id_B)^{2x} = Y^{2ex} = g^{2xye}$ with peer Bob, while Bob computes $\bar{K} = (\pi^e / id_P)^{2y} = ((XS_A g^t)^e / (id_A g^{te}))^{2y} = X^{2ey} = g^{2xye}$ but with peer P. In other words, the key is associated by Alice with Bob, while Bob associates it with P. As pointed out in [14] and many other papers, this UKS attack breaches the authentication of the protocol by binding a key to the wrong party.

3. If the session key is set to the value $\bar{K} = g^{2xye} \bmod N$ without the hash operation, then the attacker can carry a *malleability attack*. It first activates Bob in the execution of the protocol and gets (id_B, YS_B) which it transfers to Alice. She responds with (id_A, XS_A) which the attacker changes into (id_A, gXS_A) and sends it to Bob. The session key computed by Bob is $\bar{K}_B = ((gXS_A)^e / id_A)^{2y} = g^{2ey} g^{2xye}$. Now the adversary breaks into Alice's session to receive the key $\bar{K}_A = g^{2xye}$. The adversary can compute the same key $\bar{K}_B$ as Bob since $\bar{K}_B = Y^{2e} \bar{K}_A$. Note that the sessions at Alice and Bob are *not matching* (their (α, β) messages are different) and hence the security of the protocol is broken. The basic principle that keys from non-matching sessions should be computationally independent, namely, one key should leak no information on the other, is violated.

4. We need to add $id_A, id_B, \alpha, \beta$ under the hash together with $\bar{K}$, otherwise another malleability attack is possible. Alice sends $\alpha = g^x S_A$ to Bob, which the adversary intercepts and replaces with $\alpha' = (\alpha^{e+1} / A) = g^{(e+1)x} S_A$ to Bob. Bob responds with $\beta = g^y S_B$ and the adversary sends $\beta' = (\beta^{e+1} / B) = g^{(e+1)y} S_B$ to Alice. Consider now the two unmatching sessions (α, β') and (α', β): they have the same shared secret value $\bar{K} = g^{2xye(e+1)}$ and therefore the same session key, unless we add the identities and messages to the hash computation.

In addition, we have introduced one more change relative to the original Okamoto-Tanaka protocol. In Okamoto-Tanaka the key is set to g^{xye}. In our case not only is this value hashed (as discussed above) but also the computation of the secret shared value $\bar{K}$ is changed from $g^{xye} \bmod N$ to $g^{2xye} \bmod N$. We do not know of an explicit attack without this *squaring operation* but we were only able to carry the simulation in the proof with this additional squaring.

3.2 Proof of the mOT Protocol

We prove the following theorem showing the basic security of the mOT protocol. In Section 3.3 we prove further security properties, namely, resistance to KCI and to reflection attacks, and weak PFS. We defer the proof of full PFS (which requires a more involved proof and additional assumptions) to Section 4. The assumptions used in Theorem 1 are described in Section 2. The Canetti-Krawczyk security model, as specialized and extended in [25], on which we prove the protocol is described in Appendix A.

Theorem 1. *Under the RSA assumption, if we model H, H' as random oracles, the* mOT *protocol is a secure identity-based key agreement protocol.*

Proof. The proof follows a typical simulation/reduction argument: We assume an efficient KA-attacker $\mathcal{M}$ that breaks the security of the mOT protocol (see Appendix A) and use it to build an algorithm that inverts RSA on random inputs.

We start by noting the following fact about an attacker against mOT: Since the session identifier $(id_A, id_B, \alpha, \beta)$ is hashed together with the shared secret value $\bar{K}$ to obtain the session key K, we know that two different sessions necessarily correspond to two different session keys. Moreover, since the hash function H' is modelled as a random oracle then the only way for the attacker to calculate, identify, or distinguish a session key is by computing the value $\bar{K}$ and explicitly querying it from H'.[5]

We call the algorithm that we build for inverting RSA a "simulator" (denoted SIM) since it works by simulating a run of the mOT protocol against the KA-attacker $\mathcal{M}$ which is assumed to win the test-session game with non-negligible probability.

Input to SIM. The input to SIM is a triple (N, e, R) where N, e are chosen with the same RSA distribution as used in the mOT protocol and R is a random element in Z_N^*. The goal of SIM is to output $R^d \bmod N$ where d is such that $ed = 1 \bmod \phi(N)$.

Some conventions: We often omit the notation "mod$N$" when operating in the group Z_N^*. When saying that we chose a random element u from QR_N we mean chosing $v \in_R Z_N^*$ and setting $u = v^2 \bmod N$. To choose a value u in Z_N^* with a known "RSA signature" $s = u^d$, we first choose s and then set $u = s^e$. If $U = g^u, W = g^w$ are elements of QR_N we denote with $DH_g(U, W)$ the value g^{uw}, i.e. the result of the Diffie-Hellman transform in base g applied to U, W.

SIM runs a virtual execution of the mOT protocol (consisting of multiple sessions) against the attacker $\mathcal{M}$, simulating all protocol actions, including the determination of private keys and responses to queries to the functions H, H' made by $\mathcal{M}$. In particular, we allow SIM to "program" the hash functions H, H' (as long as outputs are chosen independently of each other and with uniform distribution) as is customary when modeling H, H' as random oracles.

Identities and keys. Each participant in the protocol has an identity, id_P, possibly chosen by $\mathcal{M}$; we denote $P = H(id_P)$. Of all party identities participating in the protocol, SIM chooses one at random; we denote it by id_B and will refer to this party as Bob. For each participant id_P *other than Bob*, SIM chooses a random value $p \in_R QR_N$ and sets $P = H(id_P) = p^e$. In this way, SIM also knows the private key of the participant, i.e., $S_P = P^d = p$ (note that P and S_P are elements of QR_N). For Bob, SIM sets $H(id_B) = B = R^2 \bmod N$, where R is the input to SIM (note that R^2 is random in QR_N).

Chosing a QR_N generator. SIM sets the random generator g of QR_N to be used in the protocol as following: it chooses random $\bar{r} \in_R QR_N$, sets $r = \bar{r}^e$, and $g = (rB)^e$. Note that with these choices $B = g^d/r$ and $\bar{r} = r^d$; also note that g and B are random in QR_N and independent.

Guessed test session. Before starting the simulation of session establishments, SIM chooses at random a (future) session that it conjectures will be chosen by $\mathcal{M}$ as the test session. SIM does so by guessing the holder of the test session among all the parties in the orchestrated protocol run (we refer to this party as Alice) and guessing the order number of the session among all of Alice's

[5] Note that if parties A and B have a session where they exchanged messages α (from A to B) and β (from B to A), and another session where the same messages were exchanged but in the reverse direction, both sessions will have the same key. However, as long as one of A and B is honest, each session will have at least one fresh message (except for the negligible probability that two random values in QR_N coincide), hence the above cannot happen even with the attacker's intervention (who can only choose one message in each session).

sessions. This allows SIM to know when the guessed session is activated at Alice in the protocol's run. In addition, SIM also guesses that the peer to the test session will be Bob (defined above). We specify that, if at any point in SIM's simulation, it is determined from the protocol's run that the guessed session is not to be chosen as the real test session by $\mathcal{M}$ (e.g., if either Alice or Bob are corrupted, or another test session is chosen by $\mathcal{M}$, etc.) the simulator aborts. The probability that the guessed session will actually be chosen by $\mathcal{M}$ as the test session is non-negligible (as long as the simulation of the protocol by SIM is correct).

Session Interactions (non-test sessions). Attacker $\mathcal{M}$ can choose to initiate and schedule sessions between any two participants and can input its own values into the various sessions, either by utilizing corrupted players or by delivering messages allegedly coming from honest parties. The simulator SIM needs to act on behalf of honest parties in these interactions. Simulating the actions of any uncorrupted party other than Bob is simple for SIM, as it knows their private keys and can choose their ephemeral exponents. Sessions in which Bob is a participant are more problematic since SIM does not know Bob's private key $S_B = B^d$. Whenever Bob is activated in a session, SIM will set the value $\beta = g^b/\bar{r}$ as the outgoing message from Bob where $b \in_R [1..\lfloor N/4 \rfloor]$ is chosen afresh with each activation of Bob and the value $\bar{r}$ is fixed and defined above. Clearly, β is distributed uniformly over QR_N as in the real runs of mOT. While SIM cannot compute session keys with such choice of β we will still see that it can answer the attacker's session-key queries.

Response to party corruption and session key queries (non-test sessions). If at any point $\mathcal{M}$ corrupts a party, SIM provides all information for that party including the private key (which SIM knows). Note that if the attacker asks to corrupt Bob, SIM aborts since it is a sign that SIM did not guess correctly the test session. Session-key queries for sessions where one of the messages was generated by an honest party other than Bob, can be answered by SIM who chooses the ephemeral exponent for the session. The problematic cases are sessions where Bob is a peer and for which the incoming message to Bob was chosen by the attacker (rather than by SIM itself), and provided to Bob as coming from some party id_C (we refer to it as Charlie), which may be honest or corrupted, but different than Bob.[6] In this case SIM does not know the ephemeral exponents of either party to the session so it cannot compute the session key. Instead the simulation proceeds as follows.

The idea is that as long as $\mathcal{M}$ does not query the session value $\bar{K}$ from the random oracle H', then SIM can answer the session key query with a random value. However, if $\mathcal{M}$ does know the value $\bar{K}$, and it actually queries H' on this value, then SIM needs to answer consistently. Specifically, we are dealing with a session where the peers are Bob and Charlie, whose hashed identities are $B = H(id_B)$ and $C = H(id_C)$, respectively, and the exchanged values are γ, chosen by the attacker, and β chosen by SIM as specified above. Thus, the session key is $H'(\bar{K}, id_C, id_B, \gamma, \beta)$ for the appropriately computed $\bar{K}$. Before answering the session-key query, SIM needs to check whether an input of the form $(Q, id_C, id_B, \gamma, \beta)$ was queried from H' where $Q = \bar{K}$. If such a query with $Q = \bar{K}$ was indeed performed then SIM will answer the session-key query with the existing value $H'(Q, id_C, id_B, \gamma, \beta)$. If not, SIM will choose a random value ρ in the range of H' and will return ρ as the value of the session key.

The main question is how will SIM verify whether $Q = \bar{K}$ for a prior query. The value $\bar{K}$ can be represented as $DH_g(Z^2, \beta^e/B)$ for $Z = \gamma/S_C$. By our choice of $B = g^d/r$, $\beta = g^b/\bar{r}$ and $r = \bar{r}^e$, we have that $\beta^e/B = (g^{eb}\bar{r}^{-e})/(g^d r^{-1}) = g^{eb-d}$ and therefore,

$$\bar{K} = DH_g(Z^2, \beta^e/B) = DH_g(Z^2, g^{eb-d}) = Z^{2(eb-d)} \tag{1}$$

[6] We assume for the time being that Bob does not run a KA session with itself (thus $C \neq B$). The case where both session peers have the same identity is proved in Section 3.3.

Now, since exponentiation to the e is a permutation over Z_N^*, we have that $Q = \bar{K}$ if and only if $Q^e = \bar{K}^e$, and by Equation (1) this is the case if and only if $Q^e = (Z^{2(eb-d)})^e = Z^{2(e^2b-1)}$. But this last computation can be performed by SIM who knows all the involved values, including b that SIM chose and Z (since $Z = \gamma/S_C$ and SIM knows both γ and S_C).[7]

Simulating the test session. When $\mathcal{M}$ activates the session at Alice that SIM chose as its guess for the test session, SIM acts as follows. Let the identity of Alice be id_A and denote $A = H(id_A)$. Since Alice is assumed to be the holder of the test session, it means that it is Alice (or SIM in our case) who chooses the outgoing message α from the session, not the attacker. SIM sets this message to the value $\alpha = (rB)^f S_A$, where r, B are as described at the begining of the simulation, $S_A = A^d$ is Alice's private key (which SIM knows) and f is chosen as $f = te + 1$ for $t \in_R [1..\lfloor N/4 \rfloor]$. With this choice, α's distribution is statistically close to uniform over QR_N. Indeed, we have $(rB)^{te+1} = (g^d)^{te+1} = g^d g^t$ with $t \in_R [1..\lfloor N/4 \rfloor]$ (recall that in the real protocol Alice chooses $\alpha = g^x S_A$ with g^x also statistically close to uniform distribution over QR_N). It also makes it independent of other values in the protocol including B and β.

The peer to the test session is Bob (or else SIM aborts) and the incoming message is denoted by β. This value can be chosen by the attacker (which delivers it to Alice as coming from Bob) or by Bob itself. In the latter case, β is chosen by SIM as described above for other sessions activated at Bob. In case $\mathcal{M}$ chooses β, it can be any arbitrary value. Below, we make no assumption on β other than being in Z_N^*. The session key in this case is $K = H'(\bar{K}, id_A, id_B, \alpha, \beta)$ where $\bar{K}$ is computed as follows: if $X = g^x$ denotes the value α/S_A then $\bar{K} = (\beta^e/B)^{2x}$. Now, by our choice of parameters α, g, B, r (in particular, $rB = g^d$), we have that $X = \alpha/S_A = (rB)^f = (g^d)^f$ and hence $x = df \bmod \phi(N)/4$. Thus,

$$\bar{K} = (\beta^e/B)^{2df} \tag{2}$$

Since SIM cannot compute this value (it does not know the ephemeral exponent of either peer to the session) we need to show how SIM responds to a test-session query (assuming the guessed session is indeed chosen by $\mathcal{M}$ as the test session). Upon such a query, SIM will check if there was any query made to H of the form $(Q, id_A, id_B, \alpha, \beta)$ and if so, it will check if Q equals the session value $\bar{K}$. This is done by checking whether $Q^e = (\beta^e/B)^{2f}$ (which involves values known to SIM). Indeed, note that $Q = \bar{K}$ if and only if $Q^e = \bar{K}^e$ (as exponentiation to e is a permutation) and using Equation (2) we have $\bar{K}^e = (\beta^e/B)^{2f}$. If SIM identifies such a Q, SIM has learned $\bar{K}$ from which it can compute its target RSA forgery as we explain below. If not, SIM responds to the session-key query with a random value. From now on, it monitors $\mathcal{M}$'s queries to H to see if a $Q = \bar{K}$ is identified, in which case SIM learns the session key and outputs the forgery.

Computing the forgery R^d. The goal of SIM is to compute $R^d \bmod N$ where $R \in_R Z_N^*$ was given to SIM as input. We now show that whenever SIM learns the session key corresponding to the test session (as shown above) it can compute R^d. Indeed, it is easy to see from Equation (2) that $(B^{2f})^d = \beta^{2f}/\bar{K}$, and since SIM chose $B = R^2$ then $(R^{4f})^d = \beta^{2f}/\bar{K}$. Using Lemma 1 and the fact that $4f$ is relatively prime to e we derive R^d from $(R^{4f})^d$.

Finally, we note that in order to win the test-session game with non-negligible advantage it must be that $\mathcal{M}$ queries the correct $\bar{K}$ from H with non-negligible probability, then SIM is guaranteed to learn $\bar{K}$, and hence compute R^d, also with non-negligible probability. □

[7] We note for future reference, that knowing S_C is not strictly necessary for SIM to carry this simulation step. If SIM does not know S_C (as in some other proofs in this paper) it does not know Z either. Instead SIM will use $Z^e = \gamma^e/C$ which it does know, and instead of checking $Q^e = Z^{2(e^2b-1)}$ it will check the equivalent $Q^{e^2} = (Z^e)^{2(e^2b-1)}$.

3.3 Further Security Properties of the mOT Protocol

Here we prove additional security properties of the mOT protocol that add to the protocol's appeal and advantages over other ID-based key agreement protocols (see discussion in the introduction). We prove these properties under the same assumptions used in Theorem 1.

Resistance to reflection attacks. For simplicity, in Section 3.2 we omitted the case in which sessions have the same identity at both ends, namely, when Alice and Bob have the same identity. While this situation may seem of theoretical interest only, it actually arises naturally in practice, for example when Alice's home PC establishes a secure channel with her office desktop where both devices use the same identity, say alice@company.com (hence using also the same private key). An attack that exploits the fact that the two ends use the same identity is called a reflection attack.

We now prove that the mOT protocol is secure against such attacks by adapting the proof of Theorem 1 to this case. The simulation proceeds as described in that proof except for the following two changes.

(i) In simulating responses to session-key queries for sessions where Bob is a peer and the incoming message γ is chosen by the attacker and delivered to Bob as coming from a party Charlie, we assumed that SIM knew Charlie's private key. However, if Charlie and Bob are the same, then this is not the case anymore since SIM does not know Bob's private key. Fortunately, as we remarked in footnote 7, the proof works, with a slight modification, also when SIM does not know Charlie's private key, and hence we can use it in the case of Bob = Charlie as well.

(ii) We need to consider the case that in the test session both peers, Alice and Bob, are the same. However, in the simulation of the test session in Theorem 1 we used the fact that the simulator SIM knew Alice's private key when choosing the outgoing message α. When Alice and Bob are the same, SIM does not know Alice's private key (as it does not know Bob's private key). Thus, we need to slightly change the choice of α in this case. Specifically, we now set $\alpha = (rB)^f/\bar{r}$, and hence $X = \alpha/B^d = (rB)^f/(\bar{r}B^d) = (rB)^{f-d}$ (where the last equality holds since by our choice of parameters $\bar{r} = r^d$). Also, since we have $rB = g^d$ we get that $X = g^x = (g^d)^{f-d}$ and then $x = d(f-d) \bmod \phi(N)/4$. From here $\bar{K} = (\beta^e/B)^{2x} = (\beta^e/B)^{2d(f-d)}$, and raising the two sides to the e-th power we get:

$$\bar{K}^e = (\beta^e/B)^{2(f-d)} = \beta^{2(ef-1)}B^{2d}/B^{2f} \quad \Longrightarrow \quad (B^2)^d = \frac{\bar{K}^e B^{2f}}{\beta^{2(ef-1)}}$$

which means SIM can compute $(R^4)^d = (B^2)^d$ and using Lemma 1, it can also compute the required forgery R^d.

Weak Forward secrecy. As we will prove in Section 4, the mOT protocol provides full forward secrecy even against active attackers. However, for that proof we need extra assumptions not required to prove the basic security of the protocol. So, it may be worth noting that the (much) weaker property of "weak PFS", namely, PFS against passive-only attackers can be proven solely based on RSA as in Theorem 1; actually, the hardness of factoring suffices in this case. Indeed, we show that weak PFS follows from the CDH (computational Diffie-Hellman) assumption on QR_N which is implied by the hardness of factoring [38]. The goal here is to show that if a session between uncorrupted parties Alice and Bob completes with messages α, β chosen by these parties (not chosen or modified by the attacker), then even if the attacker learns the private keys of both Alice and Bob, it will not be able to distinguish the resultant session key from a random value. Of course, in this case the attacker is not allowed to perform a session-key query against this session (which models the erasure of session keys after expiration – see Appendix A). Thus, we will be assuming

that in the test session both messages α, β, exchanged between Alice and Bob, are actually chosen by the legitimate peers and delivered unchanged by the attacker $\mathcal{M}$. However, this time we will need to be able to provide the private keys of both parties to $\mathcal{M}$ (something we could not do in the proof of Theorem 1 where the simulator SIM did not know the private key of Bob). The simulation in this case is thus very different.

The simulator SIM that we build is an algorithm that solves the CDH problem over QR_N. Its input is an RSA modulus N and a generator g of QR_N (as in the mOT protocol) as well as a pair of random elements $U = g^u, V = g^v \in_R QR_N$. The goal is to compute $W = g^{uv}$. Algorithm SIM orchestrates a run of the mOT protocol against an (assumed successful) mOT-attacker where SIM sets the parameters $(N, \bar{g}, e)$ of the mOT protocol as follows: N is the RSA modulus received by SIM as input, $\bar{g} = g^{2e}$, where g is from SIM's input, and e is chosen at random with the same probability as in the mOT protocol. In addition, SIM chooses the private keys of *all* parties, including both Alice and Bob. Thus, whenever, $\mathcal{M}$ requests to see the private keys of a party, including Alice and Bob, SIM has no problem providing them. The simulation of regular sessions is trivial (avoiding the complications in the proof of Theorem 1 stemming from the fact that in that case SIM did not know the key of Bob). When the session guessed by SIM as the test session is activated, SIM embeds the values $U = g^u$ and $V = g^v$ received as inputs as the values X and Y used to compute α and β, respectively, in that session. Note that $g^u = U = X = \bar{g}^x = g^{2ex}$ and hence $u = 2ex \bmod \phi(n)/4$; similarly, $v = 2ey \bmod \phi(n)/4$ where $Y = \bar{g}^y$.

For $\mathcal{M}$ to win the test-session game, it must query $\bar{K}$ from H where

$$\bar{K} = \bar{g}^{2exy} = (g^{2e})^{2exy} = g^{(2ex)(2ey)} = g^{uv} = CDH(U, V) = W.$$

Thus, when $\mathcal{M}$ performs this query SIM learns $\bar{K}$ and thus solves the $CDH(U, V)$ problem. Note that SIM does not even need to verify which query to H equals W as it can choose one of these queries at random and have non-negligible probability to choose the correct W.

We conclude by remarking that the forward secrecy property holds also in case of compromise of the KGC (key generation center) by which the attacker learns the private keys of all parties. That is, even after the compromise of the KGC, the attacker learns nothing about priorly exchanged session keys.

Resistance to key compromise impersonation (KCI). This is the property by which an attacker that knows the private key of Alice cannot impersonate another (uncorrupted) party to Alice. This property holds in the case of the mOT protocol. To show this property the proof of Theorem 1 remains unchanged except that now the attacker $\mathcal{M}$ can corrupt Alice; but since SIM already knows Alice's private key it has no problem furnishing this key to $\mathcal{M}$. (A formal proof uses the notion of "clean session" from [25] but the details are straightforward.)

3.4 Performance

The mOT protocol, when used with short exponents, is very efficient; in particular more efficient than any RSA-based key agreement protocol and any authenticated Diffie-Hellman protocol over Z_p^* for large prime. It is also much more efficient than any of the id-based protocols based on pairings. Yet, today's most efficient (certificate-based) key agreement protocols, such as MQV, run over elliptic curves while mOT runs over RSA composites. How do these protocols compare? In terms of operations, mOT uses one off-line and one on-line exponentiation per party (both with short exponents, say 160 or 224 bits). The e-th power computation (with $e = 3$) in mOT is essentially negligible. MQV uses the same two exponentiations plus an additional on-line "half exponentiation"

(i.e., half the number of multiplications compared to a full exponentiation). Thus, the on-line cost for MQV is 1.5 exponentiations per party compared to 1 in mOT. If in the case of MQV one uses ECDSA-signed certificates (assuming a full ECC implementation) then MQV may incur an additional cost of at least one on-line exponentiation for certificate verification (making it a total of at least 2.5 on-line exponentiations per party compared to 1 exponentiation in mOT). We have not run precise simulations but extrapolating from performance data (e.g., [10]) one can see that for moderate security parameters (say 1024-bit RSA) mOT may have a computational advantage while for larger moduli (say above 2000 bits) the advantage is on the elliptic curve side.

Regarding communication, mOT sends a single group element, say of size 2048. MQV sends a single group element, say of size 224, plus a certificate. If MQV's certificate is signed with, say, RSA-2048 then verification computation is negligible but MQV's message is larger than mOT's (especially that RSA certificates are usually significantly larger than the 2048-bit signature on them [33]). If instead the certificate is signed with ECDSA then the certificate will be shorter than RSA's but the computational complexity of verification increases by at least one exponentiation as discussed above. Finally, we recall the PFS security advantage of mOT; in MQV this can be achieved but only at the cost of a third message in the protocol.

4 Proof of the PFS Property of the mOT Protocol

In this section we prove that the Modified Okamoto-Tanaka protocol enjoys full *Perfect Forward Secrecy (PFS)* also against active attackers. Remember that weak PFS, against passive attacker only, was proven already in Section 3.3 as a direct application of the proof of Theorem 1. However, proving PFS against active attackers turns out to be more difficult. We prove this property here but for this we need to resort to two additional assumptions (on top of the RSA Assumption required for the proof of the basic security of the protocol, i.e. Theorem 1). The first assumption is the well-known "Knowledge of Exponent Assumption" introduced by Damgard [11] (and further used and studied in [17, 2]). Intuitively, it states that to compute a DH value g^{xy} out of a triple g, g^x, g^y one has to necessarily know either x or y. We will refer to this assumption (called KEA1 in [2]) as KEA-DH.

Knowledge of Exponent Assumption for Diffie-Hellman (KEA-DH). Let G be a cyclic group, and let g, h be distinct generators of G. The assumption says that for every algorithm $\mathcal{M}$ that on input G, g, h outputs (y, z) where $y = g^x$ and $z = h^x$ for some integer x, there exists an algorithm $\mathcal{M}^*$ which outputs x.

A fully formal statement of the assumption can be found in [2]; in particular, it is assumed that for every set of random coins used by $\mathcal{M}$, if $\mathcal{M}$ outputs $(y = g^x, z = h^x)$ then for the same set of random coins $\mathcal{M}^*$ outputs x.

Our second assumption is close in spirit to KEA-DH but it applies to the discrete logarithm problem; we refer to it as KEA-DL. The idea is as follows: Under the discrete log assumption, given a pair $(g, B = g^b)$ (for random b) it is hard to find b. But what if in addition to the pair $(g, B = g^b)$ one is also given a dlog oracle where one can input any value in G other than B, and receive its dlog to base g. Obviously, one can find b by querying, for example, the value Bg; more generally, one can query BV where $V = B^i g^j$ for known i, j, and compute b out of the dlog of BV. The KEA-DL assumption states that if one is allowed a single query to the oracle then the above strategy is the

only feasible one. Namely, if an algorithm finds b by querying a single value from the oracle then there is another algorithm that outputs values i, j as above. More formally[8]:

Knowledge of Exponent Assumption for Discrete Log (KEA-DL). Consider the following game between a Challenger, $Chall_{DL}$, and an adversary $\mathcal{M}$.

1. $Chall_{DL}$ provides $\mathcal{M}$ with the description of a cyclic group G, a generator g, and a random element $B = g^b$ in G;
2. $\mathcal{M}$ is allowed to query an element $V \in G$;
3. $Chall_{DL}$ responds with the discrete log of BV relative to the base g;
4. $\mathcal{M}$ outputs an integer b'; $\mathcal{M}$ wins if $b = b'$.

The KEA-DL assumption states that for every algorithm $\mathcal{M}$ that wins the above game, there exists an algorithm $\mathcal{M}^*$ which outputs integers i, j such that $V = B^i g^j$.

In our proof, we need a somewhat stronger variant of KEA-DL specific to the case of dlog over composite moduli, we refer to this variant as KEA-DL*. It essentially states that the knowledge of the e-th root of B (for a fixed value e) does not help the attacker to find the dlog of B in the above game.

Modified Knowledge of Exponent Assumption for Discrete Log (KEA-DL*). Let G be the subgroup of Quadratic Residues in Z_N^* where N is an RSA modulus. We modify the **KEA-DL** assumption to allow $\mathcal{M}$ to receive also the e-root of B (where e is an RSA exponent). The modified assumption is as follows:

1. $Chall_{DL^*}$ provides $\mathcal{M}$ with $N, e, g, B = g^b$ where g, B are random quadratic residues in Z_N^*;
2. $\mathcal{M}$ is allowed to query an element $V \in G$;
3. $Chall_{DL^*}$ responds with the discrete log of BV and the e-root of B;
4. $\mathcal{M}$ outputs an integer b'; $\mathcal{M}$ wins if $b = b'$.

The KEA-DL* assumption states that for every algorithm $\mathcal{M}$ that wins the above game, there exists an algorithm $\mathcal{M}^*$ which outputs integers i, j such that $V = B^i g^j$.

Theorem 2. *Under the RSA,* **KEA-DH** *and* **KEA-DL*** *assumptions, if we model H, H' as random oracles, then the* mOT *protocol enjoys* perfect forward secrecy (PFS) *(against passive and active attackers).*

Proof. The PFS case differs from the non-PFS case, proven in Theorem 1, in that, after completing the test session between Alice and Bob, the attacker is given the values $S_A = A^d$ and $S_B = B^d$, i.e., the private keys of the peers to the session. Only after receiving these values, the adversary needs to distinguish the test key from random.

Recall that due to the fact that the session key is the hash of the resulting secret value $\bar{K} = g^{2xye}$, where the hash is modeled as a random oracle, distinguishing the key is equivalent to finding $\bar{K}$. Thus, in the sequel we assume that a successful attacker is one that guesses this value g^{2xye}.

Examining the proof of Theorem 1 we can see that in all the simulations in that proof, the simulator knows the secret key S_A of Alice, and therefore it can provide it to the adversary upon corruption of Alice (indeed this property is what allowed us to prove resistance to KCI attacks in Section 3.3). Moreover, as shown in Section 3.3, weak PFS (i.e., against passive attackers) follows directly from the proof of Theorem 1 and requires no additional assumptions.

[8] A fully formal statement of the assumption quantifies over each set of random coins of algorithms $\mathcal{M}$ and $\mathcal{M}*$; the details are similar to the treatment of the KEA-DL assumption in [2] and are omitted from this extended abstract.

The difficulty in proving full PFS is the need to provide the attacker $\mathcal{M}$ with $S_B = B^d$ before $\mathcal{M}$ outputs its guess for the session key. This is very different than the case of Theorem 1 where the simulator first receives the attacker's guess and only then it uses this value to compute the forgery B^d. Still, with some significant changes to the simulation and some added assumptions, we will be able to prove the theorem by transforming a successful mOT attacker $\mathcal{M}$ into an RSA forger F that inverts RSA on a random input. We show this reduction now.

The RSA Forger F. The forger F is given as input an instance $N, e, R \in_R QR_N$ of the RSA problem and needs to compute $R^d \bmod N$ with non-negligible probability. F starts by running a simulation of the mOT protocol against attacker $\mathcal{M}$ (which we assume to guess the test session key with non-negligible probability). For this F sets up the public parameters of mOT as N, e, g where N, e are from F's input and $g = h^e \bmod N$ for h chosen by F at random in QR_N. As in the proof of Theorem 1, F generates private keys for all parties except Bob[9] by programming the random oracle H (i.e., for each party id_P, F chooses $p \in_R QR_N$ and sets $H(id_P) = P = p^e$, and $S_P = p$).

For Bob, F chooses the value $B = H(id_B)$ by programming H as follows. It sets a value $U = g^u \bmod N$ where F chooses u as a random integer in the range $[1..\lfloor N/2 \rfloor]$ (note that with this choice of u, the distribution of the value U is statistically close to uniform in QR_N – by a similar argument as in Section 2). Then, it flips an unbiased coin *coin*, and sets

$$H(id_B) = B = \begin{cases} R^2 \bmod N & \text{if } coin = 0 \quad \text{(where } R \text{ is part of } F\text{'s input)} \\ U^e \bmod N & \text{if } coin = 1 \end{cases}$$

Notice that the distribution of B is correct, i.e., (statistically close to) random in QR_N and independent of other values in the protocol. To simulate all the sessions other than the test session, F follows the same simulation as in Theorem 1. In other words, we keep the proof of Theorem 1 intact up to the point in that proof titled "Simulating the test session".

It remains to show how F simulates the test session interaction with $\mathcal{M}$ in the PFS case, and how this simulation results in the computation of R^d. For this we are going to first modify the attacker $\mathcal{M}$ into an attacker $\bar{\mathcal{M}}$ that behaves like $\mathcal{M}$ but, in addition, in runs where $\mathcal{M}$ guesses correctly the test session, $\bar{\mathcal{M}}$ will output some additional values that will allow F to complete its forgery. Thus, F will be running against the modified $\bar{\mathcal{M}}$ rather than against $\mathcal{M}$. We now show how we transform $\mathcal{M}$ into $\bar{\mathcal{M}}$ via several intermediate "games". We start by presenting a first game that represents the interaction with a KA-attacker in the test session experiment in the PFS setting.

The PFS Game. The following game represents the test session interaction between a "PFS challanger", $Chall_{PFS}$, and the KA-attacker $\mathcal{M}$ where $\mathcal{M}$ is allowed to corrupt Bob after the test session key is complete. In this case, $\mathcal{M}$ sends the incoming message β (allegedly coming from Bob) into the test session held by Alice, and Alice outputs a value α. After these values are set, the attacker receives Bob's secret key $S_B = B^d$ and $\mathcal{M}$ wins the game if it outputs $\bar{K} = g^{2exy}$.

PFS Game:

1. $Chall_{PFS}$ sends to $\mathcal{M}$ the values N, e, g, B, X.
   Here X represents the value α sent by Alice; indeed, since we assume that $\mathcal{M}$ can also be given S_A (which F chooses and hence knows) then providing $\alpha = X \cdot S_A$ is equivalent to providing X.

[9] Bob is the peer to the test session - as in the previous proof we assume that the simulator successfully guesses the test session and its peers, an event that happens with non-negligible probability.

2. The adversary sends a value β which in turn determines a value Y defined as $Y = \beta/B^d$. Note that β, which is chosen by $\mathcal{M}$, may not be an element of QR_N, and the same holds for Y. However, $Y^2 \bmod N$ is necessarily in QR_N and hence generated by g. Defining $y = \log_g(Y^2 \bmod N)$, and thanks to the squaring operation in the computation of the session key in mOT, we get that the session key g^{2xye} equals X^{ye}.
3. $Chall_{PFS}$ sends B^d.
4. $\mathcal{M}$ sends $\bar{K}$. The adversary, $\mathcal{M}$, wins if $\bar{K} = X^{ye}$.

We now show how to transform an atacker $\mathcal{M}$ that wins the above PFS game (with non-negligible probability) into a modified attacker $\bar{\mathcal{M}}$ (with the same probability of success) that F will use to compute its RSA forgery. We arrive to $\bar{\mathcal{M}}$ from $\mathcal{M}$ through a series of adversaries $(\mathcal{M}, \mathcal{M}^*, \mathcal{M}_1, \mathcal{M}_1^*, \bar{\mathcal{M}})$ defined in the following games.

The adversary $\mathcal{M}^*$: Note that $\mathcal{M}$, in the above PFS game, can be changed to also output Y^{2e} (computed as β^{2e}/B^2). So in a winning run on input g, X, attacker $\mathcal{M}$ would output $\bar{K} = X^{ye}$ and also $Y^{2e} = g^{ye}$. By invoking the **KEA-DH** assumption there is another attacker $\mathcal{M}^*$ that behaves as $\mathcal{M}$ but in addition it outputs, in winning runs, $ye = \log_g Y^{2e}$ in step 4.

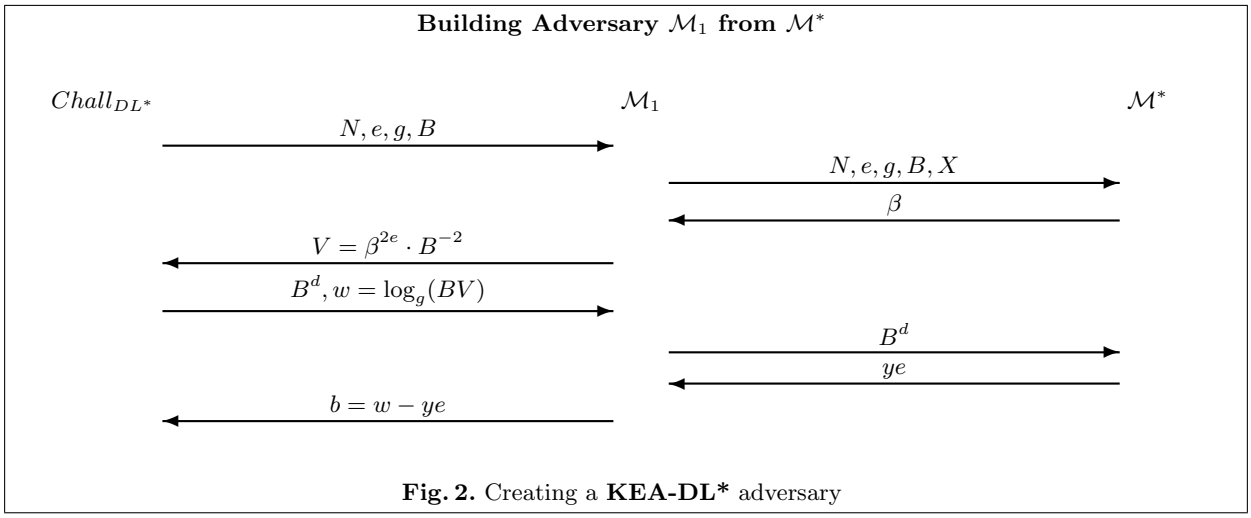

Fig. 2. Creating a **KEA-DL*** adversary

The adversary $\mathcal{M}_1$: We now use the modified PFS attacker $\mathcal{M}^*$ to build an attacker $\mathcal{M}_1$ that interacts with a challenger $Chall_{DL^*}$ in a **KEA-DL*** game. The actions of $\mathcal{M}_1$ are described in Figure 2: $\mathcal{M}_1$ uses $\mathcal{M}^*$ as a subroutine (where $\mathcal{M}_1$ acts as the PFS challenger with respect to $\mathcal{M}^*$) and uses responses received from $\mathcal{M}^*$ to answer $Chall_{DL^*}$ queries. Now, since in a successful run of $\mathcal{M}^*$ the last value ye output by $\mathcal{M}^*$ satisfies $g^{ye} = Y^{2e} = \beta^{2e}/B^2 = V$, and the value w output by $Chall_{DL^*}$ satisfies $w = \log_g BV = b + ye$, then the value $b = w - ye$ answered by $\mathcal{M}_1$ is correct (i.e., $g^b = B$) and $\mathcal{M}_1$ wins the KEA-DL* game. In other words, each successful run of $\mathcal{M}^*$ (which happens with non-negligible probability) induces a successful run of $\mathcal{M}_1$ in the **KEA-DL*** game.

The adversary $\mathcal{M}_1^*$: By the **KEA-DL*** Assumption, there is an adversary $\mathcal{M}_1^*$ that behaves exactly as $\mathcal{M}_1$ except that together with V it also outputs i, j such that $V = B^i g^j$. Replacing $\mathcal{M}_1$ with $\mathcal{M}_1^*$ in Figure 2, we get the same flows except that now the values i, j as above are added to the second flow from $\mathcal{M}_1$ to $Chall_{DL*}$. We refer to this as the *modified game of Figure 2.*

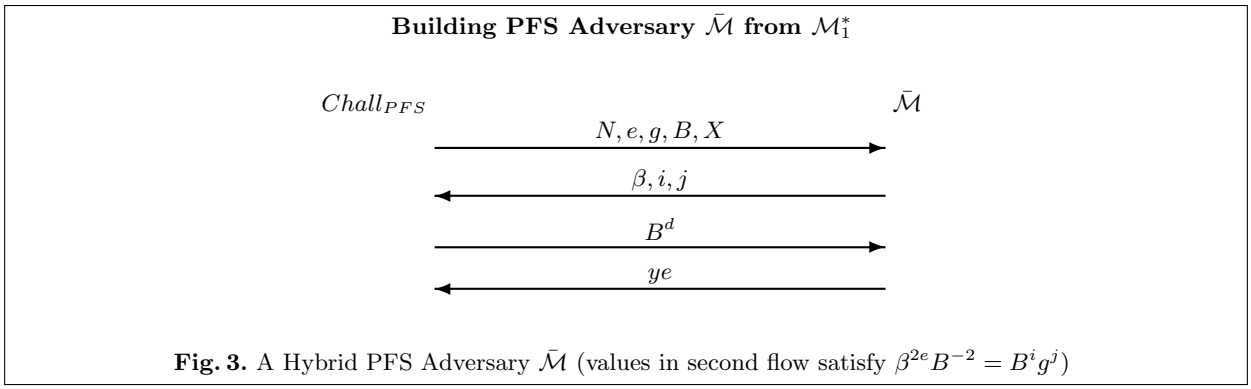

Fig. 3. A Hybrid PFS Adversary $\bar{\mathcal{M}}$ (values in second flow satisfy $\beta^{2e}B^{-2} = B^i g^j$)

The hybrid adversary $\bar{\mathcal{M}}$: Using $\mathcal{M}_1^*$ we build an attacker $\bar{\mathcal{M}}$ that interacts in a PFS game as represented in Figure 3. In the first flow $\bar{\mathcal{M}}$ receives inputs from $Chall_{PFS}$ as in a regular PFS game. Next $\bar{\mathcal{M}}$ uses these inputs to call $\mathcal{M}_1^*$ in the modified game of Figure 2. In this modified game, $\mathcal{M}_1^*$ uses these same values as its first flow to $\mathcal{M}^*$. Upon receiving the β response from $\mathcal{M}^*$, $\mathcal{M}_1^*$ produces the value $V = \beta^{2e}B^{-2}$ as well as i, j such that $V = B^i g^j$. Then, $\bar{\mathcal{M}}$ sends to $Chall_{PFS}$ the values β, i, j (note that i, j are not part of the basic PFS game but additional outputs produced by $\bar{\mathcal{M}}$). In the next flow, $\bar{\mathcal{M}}$ receives from $Chall_{PFS}$ the value B^d as in a PFS game, which $\bar{\mathcal{M}}$ uses as the third flow from $\mathcal{M}_1^*$ to $\mathcal{M}^*$. Finally, upon receiving the response ye from $\mathcal{M}^*$, $\bar{\mathcal{M}}$ outputs this same value as its fourth message in the PFS game.

It is important to observe that the flows involving the value w produced by the dlog oracle in Figure 2 are *not* used by $\bar{\mathcal{M}}$; this is important since in a real execution of the PFS game – as part of the interaction with a mOT attacker – there is no such dlog oracle (this only exists as an artifact of the proof).

We note that in a successful run of $\bar{\mathcal{M}}$, the values β, i, j, ye output by $\bar{\mathcal{M}}$ satisfy the following equations:

$$g^{ye} = Y^{2e} = (\beta^e / B)^2 \bmod N \tag{3}$$

$$\beta^{2e}B^{-2} = V = B^i g^j \bmod N \tag{4}$$

which yield

$$g^{ye} = B^i g^j \bmod N \tag{5}$$

Also note that by virtue of the above sequence of games, if $\mathcal{M}$ succeeds with non-negligible probability then $\bar{\mathcal{M}}$ succeeds in outputting the above values also with non-negligible probability.

Summarizing the above transformations, we showed that the existence of a successful PFS attacker $\mathcal{M}$ implies (via the KEA assumptions) the existence of a second PFS attacker $\bar{\mathcal{M}}$ that in addition to the normal outputs of $\mathcal{M}$ (i.e., β and the session key guess), it also outputs the values i, j, ye that satisfy the above equations.

Applying the above to the key agreement setting of mOT we get that one can take a KA-attacker $\mathcal{M}$ that successfully breaks the PFS property of mOT and transform it into an equivalent KA-attacker $\bar{\mathcal{M}}$, that differs from $\mathcal{M}$ in that it also outputs the above values i, j, ye during a successful interaction in the test session experiment. We use this modified KA-attacker $\bar{\mathcal{M}}$ to complete the description of the forger F.

Back to the RSA Forger F. We have seen before how F simulates a run of mOT against a KA-attacker $\mathcal{M}$ except for the test session interaction. Here we complete the description of this

part of the simulation and show how F computes its RSA forgery. For this we consider a run of F (as described so far) against the modified KA-adversary $\bar{\mathcal{M}}$ defined above; that is, $\bar{\mathcal{M}}$ behaves exactly as $\mathcal{M}$ but, in successful runs of $\mathcal{M}$, in addition to β it outputs i, j and in addition to the guess of the session key it outputs ye.

The way F uses $\bar{\mathcal{M}}$ to generate a forgery depends on two values: whether i output by $\bar{\mathcal{M}}$ is 0 or not, and whether the *coin* chosen by F (see above) is 0 or 1. We now show these different cases.

Case $i = 0$. With probability 1/2 we also have that $coin = 0$ and therefore $B = R^2 \bmod N$. In this case, F is running $\bar{\mathcal{M}}$ on $g = h^e$, $B = R^2$ and $X \in_R QR_N$. Assuming the run of $\bar{\mathcal{M}}$ is successful and $i = 0$, the forger F obtains the value $j = ye \bmod p'q'$ in the second message from $\bar{\mathcal{M}}$ in Figure 3 (recall that by (5) we have $ye = ib + j \bmod p'q'$). F then uses j to compute Y^2 as follows:

$$h^j = h^{ye} = (g^d)^{ye} = g^y = Y^2 \bmod N,$$

and uses Y^2 to compute $(R^4)^d = B^{2d} = (\beta^2/Y^2) \bmod N$. Using Lemma 1 and the fact that 4 and e are relatively prime F can compute, out of the value $(R^4)^d$, the required forgery $R^d \bmod N$.

Important: In the above case, F only needs to know j to complete its forgery; therefore it does not need to complete the third and fourth flows in Figure 3 (that involve B^d) before it can forge.

Case $i \neq 0$. With probability 1/2 we also have that $coin = 1$ and therefore $B = U^e = g^{ue} \bmod N$ where, by definition, u is a random integer in the range $[1..\lfloor N/2 \rfloor]$. In this case, F knows $B^d = U$ and hence can complete the full run against $\bar{\mathcal{M}}$ in Figure 3. That is, F runs $\bar{\mathcal{M}}$ sending (g, B, X) in the first message and U in the third message. F receives from $\bar{\mathcal{M}}$ the values i, j in the second message and $y' = ye$ in the last message. F will use i, j, y' to find two values s and t from which it will derive (whp) the factorization of N.

Specifically, F sets $s = ie$ and $t = y' - j$ (these operations are over the integers). It holds $(\bmod\ p'q')$ that

$$t = y' - j = ye - j = ib = ieu = su \bmod p'q'$$

where the third equality is from Equation (5) and $b = eu \bmod p'q'$ holds by the choice $B = U^e = g^{ue}$.

Thus, the integer $t - su$ is a multiple of $p'q'$ that F can compute as it knows s, t, u. If this number is non-zero then F factors N (as it is well known, the factorization of N can be found from such a multiple of $p'q'$, e.g. [30]). Now, note that there are at least two values of the integer u that will result in the same element U in QR_N (since $u \in_R [1..\lfloor N/2 \rfloor]$ and $\lfloor N/2 \rfloor \geq 2p'q'$). Since $\bar{\mathcal{M}}$ knows U but not the specific u (indeed, from the value U one cannot learn which of the possible values of u was chosen by F), the probability that t and s (that are derived from $\bar{\mathcal{M}}$ outputs) result in $t - su$ being zero is at most 1/2 (at most one of the u's can satisfy this equation).

In all, we have that if F interacts with a successful run of $\mathcal{M}$ (which implies a successful run of $\bar{\mathcal{M}}$) and $i = 0$ then with probability 1/2 F correctly computes R^d. If the run of $\mathcal{M}$ is successful and $i > 0$ then with probability 1/2 F factors N and hence can produce R^d as well. Since $\mathcal{M}$ (and $\bar{\mathcal{M}}$) succeed with non-negligible probability so does F in computing the forgery $R^d \bmod N$. □

5 A Multi-Authority Extension of mOT and Its Security

Schridde, Smith and Freisleben [37] show how to extend the Okamoto-Tanaka protocol to the multi-authority (or multi-domain) case, in which users from two separate and independent key generation centers (or trusted authorities) can still establish a secret key. The original protocol as proposed in [37] has the same vulnerabilities as the mOT protocol which arise from the lack of hashing on

the session key. Our description below incorporates the required modification to ensure the security and proof of the protocol. We also make a change to improve its efficiency. In the rest of the paper we will refer to this as the SSF protocol.

Setup Phase. Consider two trusted authorities TA_1 and TA_2 each set up as in the Okamoto-Tanaka system. TA_i has public parameters $(N_i = p_i q_i, e_i, g_i)$ and secret key d_i, s.t. $e_i d_i = 1 \bmod \phi(N_i)$. A party (say) Alice which belongs to TA_i receives her secret key $S_A = A^{d_i} \bmod N_i$ where $A = H(id_A)$. We assume that the identities of the parties include a description of the TA they belong to (e.g. in this case id_A would include N_i, e_i, g_i). In the following, Alice and Bob are two parties from TA_1 and TA_2, respectively, and they want to generate a shared key. We assume the TA_i have chosen safe primes in the generation of their moduli, i.e. we assume that $p_i = 2p'_i + 1$ and $q_i = 2q'_i + 1$ with p'_1, p'_2, q'_1, q'_2 all primes. In the following we denote with $\phi_i = p'_i q'_i$, the order of the quadratic residue subgroup modulo N_i.

The Key Exchange

Let $E = lcm(e_1, e_2)$. Both parties compute the following values $\bmod N_1 N_2$ using the Chinese Remainder Theorem:

1. $\hat{g} = \begin{cases} g_1 \bmod N_1 \\ g_2 \bmod N_2 \end{cases}$
2. $\hat{A} = \begin{cases} A^{E/e_1} \bmod N_1 \\ 1 \bmod N_2 \end{cases}$
3. $\hat{B} = \begin{cases} 1 \bmod N_1 \\ B^{E/e_2} \bmod N_2 \end{cases}$
4. Alice and Bob locally (and secretly) compute the following values $\bmod N_1 N_2$, respectively,
   $\hat{S_A} = \begin{cases} S_A \bmod N_1 \\ 1 \bmod N_2 \end{cases}$ and $\hat{S_B} = \begin{cases} 1 \bmod N_1 \\ S_B \bmod N_2 \end{cases}$

Alice chooses at random an integer x in a set of integers S (to be specified below) computes $\hat{X} = \hat{g}^x \bmod N_1 N_2$ and transmits $\hat{\alpha} = \hat{X}\hat{S_A} \bmod N_1 N_2$ to Bob.

Bob similarly chooses at random an integer y in the same set S, computes $\hat{Y} = \hat{g}^y \bmod N_1 N_2$ and transmits $\hat{\beta} = \hat{Y}\hat{S_B} \bmod N_1 N_2$ to Alice.

Alice computes the shared secret value:

$$\bar{K_A} = \left(\frac{\hat{\beta}^E}{\hat{B}}\right)^{2x} = \left(\frac{\hat{Y}^E \hat{S_B}^E}{\hat{B}}\right)^{2x} = \hat{g}^{2xyE} \left(\frac{\hat{S_B}^E}{\hat{B}}\right)^{2x} = \hat{g}^{2xyE} \bmod N_1 N_2$$

Indeed the value $\frac{\hat{S_B}^E}{\hat{B}}$ is equal to 1 both $\bmod N_1$ (as $\hat{S_B} = \hat{B} = 1 \bmod N_1$) and $\bmod N_2$ as

$$\frac{\hat{S_B}^E}{\hat{B}} = \frac{S_B^E}{B^{E/e_2}} = \frac{B^{d_2 e_2 (E/e_2)}}{B^{E/e_2}} = \frac{B^{E/e_2}}{B^{E/e_2}} = 1 \bmod N_2$$

Alice hashes the secret value together with the ids of the parties and the exchanged messages to obtain the session key $K = H'[\bar{K_A}, id_A, id_B, \hat{\alpha}, \hat{\beta}]$.

Bob computes his key in the symmetric manner, i.e.

$$\bar{K_B} = \left(\frac{\hat{\alpha}^E}{\hat{A}}\right)^{2y} = \hat{g}^{2xyE} \bmod N_1 N_2 \quad \text{and} \quad K = H'\left[\bar{K_B}, id_A, id_B, \hat{\alpha}, \hat{\beta}\right]$$

The set S of ephemeral exponents. The choice of the set S follows similar criteria as in the case of the basic mOT protocol (Section 3). The order of the element $\hat{g}$ is $p'_1p'_2q'_1q'_2$ which is closely approximated by $\lfloor N_1N_2/16 \rfloor$, so by choosing S to be the integers in the interval $[1..\lfloor N_1N_2/16 \rfloor]$ one generates a distribution statistically close to uniform over $\langle \hat{g} \rangle$. This can be improved, as in the case of the basic mOT protocol and following [18, 15], by choosing the much smaller interval $[1..\lfloor \sqrt{N_1N_2}/4 \rfloor]$ which results in a distribution that is computationally indistinguishable from uniform over $\langle \hat{g} \rangle$. However, also as in the basic mOT case and following [15], if one assumes that over Z_N^* discrete log is hard for exponents of length κ, then the SSF protocol is secure even if S is chosen as the set of integers of length κ (typically, one would choose κ to be 160 or 224). We recommend this choice of S in practice.

Modifications to the original protocol from [37]. As in the case of the mOT protocol of Section 3, also here modifications to the original protocol from [37] were required in order to ensure the security of the protocol and its proof. One modification, similar to the mOT case, is the computation of the session key via hashing the shared value $\bar{K}_A = \bar{K}_B$ together with the session information using H'. This change is mandatory for the same reasons explained in Section 3.1 for the basic mOT protocol (in particular the malleability attacks described in (3) and (4) can be carried out as well in this case). We note that the hashing of identities using H, when computing the secret keys, was already incorporated to the protocol description in [37] and hence that part did not require modification.

Another change we introduce is to use E as the exponent to compute the session key, while in [37] the product e_1e_2 is used. Our choice clearly improves performance, as in general E can be much smaller than the product.

Finally two more changes were made to the original protocol from [37]: (i) in [37] the value $\hat{g}$ is computed as $g_1 \cdot g_2 \bmod N_1N_2$ while we use the CRT to define $\hat{g}$; (ii) the value of the secret value shared by the parties before hashing is squared, i.e. $\hat{g}^{2Exy}$ while in [37] is $\hat{g}^{e_1e_2xy}$. We are not aware of any attacks if one does not incorporate these last two modifications, but we were able to devise a proof only with them.

5.1 Proof of the SSF Protocol

Theorem 3. *Under the RSA Assumption, if we model H, H' as random oracles, then protocol* SSF *is a secure identity-based key agreement protocol.*

The proof of the SSF protocol follows the overall structure of the proof of the mOT protocol in Section 3.2 with a few technical careful modifications.

We build the simulator as an RSA forger SIM that on input an RSA problem N, e, R computes the value $R^d \bmod N$.

SIM orchestrates a run of the SSF protocol. For TA_2 the simulator sets $N_2 = N, e_2 = e$ and chooses a value g_2 in a way we specify below. For TA_1 the simulator chooses the parameters N_1 (including its factorization), e_1 and g_1.

With this choice of parameters SIM can compute the secret keys of all parties in TA_1. For the parties in TA_2, the simulator chooses a random party Bob with identity id_B and sets $B = H(id_B) = R^2 \bmod N_2$. For any party, other than Bob, with identity id_P the simulator chooses $p \in_R QR_{N_2}$ and sets $H(id_P) = P = p^{e_2} \bmod N_2$: this way the simulator knows the secret key $S_P = p$ of id_P in TA_2.

As in the proof of Theorem 1 the simulator chooses a random quadratic residue $\bar{r} \bmod N_2$ and sets $r = \bar{r}^{e_2} \bmod N_2$, $g_2 = (rB)^{e_2} \bmod N_2$. If $d_2 = e_2^{-1} \bmod \phi(N_2)$ then $rB = g_2^{d_2}$ and $\bar{r} = r^{d_2}$.

The simulator guesses the test session as a random one involving Bob. As in the proof of Theorem 1 the simulator can perfectly simulate all the sessions that do not involve Bob. We need to show how to simulate the sessions at Bob.

First we simulate sessions different from the guessed test session. Let Charlie be a party from TA_1 with identity id_C (the case in which Charlie is from TA_2 can be handled as in the case of the mOT protocol proof) who opens a session with Bob. If Charlie is uncorrupted, this means that during the simulation SIM chooses Charlie's messages – if Charlie sends out $\hat{\gamma} = \hat{g}^z \hat{S_C}$ then SIM chooses z. When SIM chooses the message from Bob, it generates the message, $\hat{\beta}$ as a random element in the group $\langle \hat{g} \rangle$. Note that $\hat{\beta} = \hat{g}^y \hat{S_B}$ for some y and $\hat{S_B}$ which F does not know, but he can calculate $\hat{g}^{yE}$ as $\hat{\beta}^E / \hat{B}$. Thus, if this session key is queried by the attacker, then F can respond with $H'[(\hat{\beta}^E/\hat{B})^{2z}, id_C, id_B, \hat{\gamma}, \hat{\beta}]$ (since it knows z).

A more serious problem with the simulation is with sessions at Bob that were generated by the attacker by presenting an incoming session message $\hat{\gamma}$ to Bob as coming from Charlie but not generated by SIM. In particular we do not even know if the value $\hat{\gamma}$ is in the group generated by $\langle \hat{g} \rangle$. In this case, how will SIM respond to a session key query for that session? SIM will proceed as follows: It computes $Z = \hat{\gamma}/\hat{S_C} \bmod N_1 N_2$. For Bob's response the simulator chooses $b_1 \in_R [1..\lfloor N/4 \rfloor 0]$ and $b_2 \in_R [1..\lfloor N_2/4 \rfloor]$ and using the Chinese Remainder Theorem sets Bob's response $\hat{\beta} \bmod N_1 N_2$ as

$$\hat{\beta} = \begin{cases} \beta_1 = g_1^{b_1} \bmod N_1 \\ \beta_2 = g_2^{b_2}/\bar{r} \bmod N_2 \end{cases}$$

The shared secret value of this session is then $\bar{K} = DH_{\hat{g}}(Z^2, \hat{\beta}^E/\hat{B})$ and the session key will be $K = H'[\bar{K}, id_C, id_B, \hat{\gamma}, \hat{\beta}]$. The simulator therefore checks if for any query by the attacker to the random oracle H' of the form $[Q, id_C, id_B, \hat{\gamma}, \hat{\beta}]$ we have that $Q = \bar{K}$ (in a way described below). If such a query is found then SIM answer with whatever answer was provided in the first place, otherwise it answers at random and then for any future query of the form $[Q, id_C, id_B, \hat{\gamma}, \hat{\beta}]$ it checks if $Q = \bar{K}$ for consistency.

We check if $Q = \bar{K} = DH_{\hat{g}}(Z^2, \hat{\beta}^E/\hat{B})$ by checking separately modN_1 and modN_2. Indeed (recall that $\hat{B} = 1 \bmod N_1$ and $\hat{B} = B^{E/e_2} \bmod N_2$) $Q = DH_{\hat{g}}(Z^2, \hat{\beta}^E/\hat{B}) \bmod N_1 N_2$ if and only if $Q = DH_{g_1}(Z^2, g_1^{b_1 E}) \bmod N_1$ and $Q = DH_{g_2}(Z^2, (g_2^{b_2}/\bar{r})^E / B^{E/e_2}) \bmod N_2$.

The first test modN_1 can be easily performed since $Q = DH_{g_1}(Z^2, g_1^{b_1 E}) \bmod N_1$ if and only if $Q = Z^{2b_1 E} \bmod N_1$. The second test modN_2 can also be performed as follows: recall that $B = g^{d_2}/r = g^{d_2}/\bar{r}^{e_2} \bmod N_2$, therefore $B^{E/e_2} = g^{d_2 E/e_2}/\bar{r}^E \bmod N_2$, therefore it must hold that

$$Q = DH_{g_2}(Z^2, g_2^{b_2 E - d_2 E/e_2}) \bmod N_2$$

which can be verified by checking that (recall that exponentiating to the e_2 power is a permutation modN_2) $Q^{e_2} = Z^{2(b_2 E e_2 - E/e_2)} \bmod N_2$ which can be computed by SIM since it has all the necessary values.

Simulation of the test session. We now consider the simulation for the guessed test-session between Alice and Bob. We assume Alice is in TA_1 (otherwise the proof can be carried out as in Theorem 1). Here too we have two cases: if the message $\hat{\beta}$ has been produced by the adversary (who sends it to Alice as if coming from Bob) or by Bob itself (in which case SIM produces it during the simulation). Either case we assume $\hat{\beta}$ to be an arbitrary element mod$N_1 N_2$.

Recall that SIM knows the secret key S_A for Alice. Thus SIM can choose the element $\hat{\alpha} = \hat{X}\hat{S_A} \bmod N_1N_2$ in the test session by choosing $\hat{X}$ using the Chinese Remainder Theorem as

$$\hat{X} = \begin{cases} g_1^{a_1} \bmod N_1 \\ (rB)^f = g^{d_2 f} \bmod N_2 \end{cases}$$

where $f = te_2 + 1$ for $t \in_R [1..\lfloor N_2/4 \rfloor]$. As in the proof of Theorem 1 the value $\hat{\alpha}$ follows a distribution which is statistically close to the one of the real protocol, and is independent of all the other values.

First we note that the shared secret value of this session $\bmod N_2$ is equal to

$$\bar{K} = \left[\frac{\hat{\beta}^E}{B^{E/e_2}}\right]^{2d_2 f} \bmod N_2$$

Assuming that SIM can find $\bar{K}$ we have (recall that $B = R^2 \bmod N_2$)

$$(R^{4fE/e_2})_2^d = \frac{\hat{\beta}^{2fE/e_2}}{\bar{K}}$$

and using Lemma 1 we get the value $R^{d_2} \bmod N_2$ as desired (notice indeed that $4fE/e_2$ is relatively prime to e_2).

To find $\bar{K}$ we observe that if the adversary wins the test session game, i.e. distinguishes the real test session $K = H'[\bar{K}, id_A, id_B, \hat{\alpha}, \hat{\beta}]$ from a random one, then it must have queried the random oracle H' on $[\bar{K}, id_A, id_B, \hat{\alpha}, \hat{\beta}]$. One possibility is to choose a random query of the form $[Q, id_A, id_B, \hat{\alpha}, \hat{\beta}]$ and set $Q = \bar{K}$ (this will be correct with non-negligible probability).

But we can do better, indeed the correct query can be recognized by SIM since among all queries of the form $[Q, id_A, id_B, \hat{\alpha}, \hat{\beta}]$, for the correct one it holds that

$$Q^{e_2} = \left[\frac{\hat{\beta}^E}{B^{E/e_2}}\right]^{2f} \bmod N_2$$

and

$$Q = \left[\frac{\hat{\beta}^E}{B^{E/e_2}}\right]^{2a_1} \bmod N_1$$

6 Adaptation of the mOT Protocol to the Certificate-based Setting

In this section we outline an adapttion of the mOT and SSF protocols to the non-ID-based settings, i.e., the traditional setting in which each user chooses a pair of public/secret keys and has it certified by a certification authority. As we pointed out in the Introduction, the outstanding efficiency of the mOT protocol makes it an attractive candidate for key agreement also in this setting.

A proposal based on the mOT protocol This proposal requires a trusted key generation center (KGC) to properly set up a common modulus N for all the users as well as a public exponent e and a generator g. Once these parameters are chosen and published, the KGC can erase any sensitive information (i.e., the factorization of N and private exponent d) and self-destroy. Indeed, users are able to choose their own (random) private/public key pairs by themselves just using the public parameters (this is reminiscent of the Fiat-Shamir and Guillou-Quisquater schemes). Thus,

this scheme puts significantly weaker trust on the KGC compared to ID-based trusted authorities which have to maintain the master secret key for the entire life of the system in order to give secret keys to new users.[10] The details are described below.

Set-up Phase: A trusted KGC computes N as the product of two safe primes, i.e. $N = (2p' + 1)(2q' + 1)$ with p', q' primes. It also chooses an element g of order $p'q'$ and an integer e relatively prime to $\phi(N)$. It publishes N, e, g, safely erases the factors of N and self-destroys (stops operation)

User Key Generation. On input N, e, g the user Alice chooses a random number $r \in Z_N^*$ and sets the secret key $S_A = r^2 \bmod N$ and its public key $A = S_A^e \bmod N$. The public key A is certified in the usual manner by a CA.

Key Agreement. Users Alice and Bob, with public keys A, B and secret keys S_A, S_B respectively run the mOT protocol (using the public keys instead of the identities, and possibly exchanging public-key certificates). Computation and communication costs are exactly as in the id-based mOT protocol.

A KGC-free proposal based on SSF We can use the SSF protocol to *completely remove* the need for a trusted set-up of the system (no KGC required, just a regular CA to issue PK certificates). The idea is to have each party choose its own modulus (which eliminates the trusted KGC role), and then leverage on the multi-domain feature of the SSF protocol. Here are the details.

User Key Generation. Alice chooses an RSA modulus N_A as the product of two safe primes, i.e. $N_A = (2p'_A + 1)(2q'_A + 1)$ with p'_A, q'_A primes. It also chooses an element g_A of order $p'_A q'_A$ and an integer e_A relatively prime to $\phi(N_A)$. Finally it chooses a random number $r_A \in Z_N^*$ and sets $S_A = r_A^2 \bmod N_A$ and $A = S_A^e \bmod N_A$. The public key is $[N_A, g_A, e_A, A]$ (certified in the usual manner by a CA after receiving proof by Alice that she knows the secret value r_A) and the secret key is S_A.

Key Agreement. Users Alice and Bob, with public keys $[N_A, g_A, e_A, A]$, $[N_B, g_B, e_B, B]$ and secret keys S_A, S_B respectively run the SSF protocol using N_A, g_A, e_A and N_B, g_B, e_B as the parameters of the two TA's, and A, B as the identities.

Performance. Since Alice chooses her own modulus $N_A = p_A q_A$ (and hence she knows p_A, q_A), she can perform the computations modulo $N_A N_B$ via the Chinese Remainder Theorem on the values p_A, q_A, N_B. This results in a mere 25% increase in the computation cost relative to the basic mOT protocol but requires twice as many bits.

Acknowledgment. Research was sponsored by US Army Research laboratory and the UK Ministry of Defense and was accomplished under Agreement Number W911NF-06-3-0001. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the US Army Research Laboratory, the U.S. Government, the UK Ministry of Defense, or the UK Government. The US and UK Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.

[10] Sometimes KGCs are justified on the basis of providing key escrow or key recovery services; in our scheme this can be done if the KGC is kept active for such purpose, but even in this case the KGC does not need to be involved with the production of private keys on behalf of users.

References

1. Ben Adida, Susan Hohenberger, and Ronald L. Rivest, "Lightweight Email Signatures", 2005.
2. M. Bellare and A. Palacio, "The Knowledge-of-Exponent Assumptions and 3-round Zero-Knowledge Protocols", *Crypto'04*, LNCS 3152.
3. Blom, R. 1985. An optimal class of symmetric key generation systems. In Eurocrypt 84.
4. Colin Boyd, Kim-Kwang Raymond Choo: Security of Two-Party Identity-Based Key Agreement. Mycrypt 2005: 229-243
5. Colin Boyd, Wenbo Mao, Kenneth G. Paterson: Key Agreement Using Statically Keyed Authenticators. ACNS 2004: 248-262
6. R. Canetti, H. Krawczyk: Analysis of Key-Exchange Protocols and Their Use for Building Secure Channels. EUROCRYPT 2001: 453-474
7. R. Canetti, H. Krawczyk: Universally Composable Notions of Key Exchange and Secure Channels. EUROCRYPT 2002: 337-351
8. L. Chen, Z. Cheng, Nigel P. Smart: Identity-based key agreement protocols from pairings. Int. J. Inf. Sec. 6(4): 213-241 (2007)
9. L. Chen and C. Kudla. Identity Based Authenticated Key Agreement Protocols from Pairings In 16th IEEE Computer Security Foundations Workshop - CSFW 2003, pages 219-233. IEEE Computer Society Press, 2003.
10. Wei Dai. Crypto++ 5.5 Benchmarks. `http://www.cryptopp.com/benchmarks.html`, September 2009.
11. I. Damgård, "Towards Practical Public Key Systems Secure Against Chosen Ciphertext Attacks", Crypto'91, LNCS Vol. 576.
12. A. De Santis, Y. Desmedt, Y. Frankel and M. Yung. How to share a function securely. STOC '94: pp.522–533, ACM Press.
13. W. Diffie and M. Hellman, "New Directions in Cryptography", IEEE Trans. Info. Theor. 22, 6 (Nov 1976), pp. 644–654.
14. W. Diffie, P. van Oorschot and M. Wiener, "Authentication and authenticated key exchanges", *Designs, Codes and Cryptography*, 2, 1992, pp. 107–125.
15. O. Goldreich and V. Rosen. On the security of modular exponentiation with application to the construction of pseudorandom generators. Journal of Cryptology, 16(2):71-93 (2003).
16. Gunther, C.G.: An Identity-Based Key-Exchange Protocol. EUROCRYPT 1989. LNCS, vol. 434, pp. 29-37.
17. Hada, S. and Tanaka, T., "On the Existence of 3-round Zero-Knowledge Protocols", Crypto'98, LNCS Vol. 1462.
18. J. Hastad, A. Schrift, A. Shamir, "The Discrete Logarithm Modulo a Composite Hides O(n) Bits," *J. Comput. Syst. Sci.*, 47(3): 376-404 (1993)
19. Dennis Hofheinz and Eike Kiltz, "The Group of Signed Quadratic Residues and Applications", CRYPTO 2009,
20. ISO/IEC IS 9798-3, "Entity authentication mechanisms — Part 3: Entity authentication using asymmetric techniques", 1993.
21. Antoine Joux: A One Round Protocol for Tripartite Diffie-Hellman. J. Cryptology 17(4): 263-276 (2004). Also in ANTS 2000.
22. S. Kim, M. Mambo, T. Okamoto, H. Shizuya, M. Tada, and D. Won. On the Security of the Okamoto-Tanaka ID-based Key Exchange Scheme against Active Attacks. IEICE Transactions Fundamentals, E84-A(1):231-238, January 2001.
23. H. Krawczyk: SKEME: A Versatile Secure Key Exchange Mechanism for Internet. 1996 Internet Society Symposium on Network and Distributed System Security (NDSS), (1996).
24. H. Krawczyk: SIGMA: The 'SIGn-and-MAc' Approach to Authenticated Diffie-Hellman and Its Use in the IKE-Protocols. CRYPTO 2003: 400-425
25. Hugo Krawczyk: HMQV: A High-Performance Secure Diffie-Hellman Protocol. CRYPTO 2005: 546-566
26. L. Law, A. Menezes, M. Qu, J. Solinas, and S. Vanstone, "An efficient Protocol for Authenticated Key Agreement", *Designs, Codes and Cryptography, 28, 119-134, 2003.*
27. Masahiro Mambo and Hiroki Shizuya, "A Note on the Complexity of Breaking Okamoto-Tanaka ID-Based Key Exchange Scheme," IEICE Trans. Fundamentals, Vol.E82-A, No.1 (1999), pp.77-80. (Also in PKC'98.)
28. U. Maurer and S. Wolf, "Diffie-Hellman oracles", CRYPTO '96, LNCS, vol. 1109, 1996, pp. 268-282.
29. N. McCullagh and P. S. L. M. Barreto. A New Two-Party Identity-Based Authenticated Key Agreement In Cryptographers Track at RSA Conference - CT-RSA 2005, LNCS 3376.
30. A. Menezes, P. Van Oorschot and S. Vanstone, "Handbook of Applied Cryptography," CRC Press, 1996.
31. E. Okamoto. Key Distribution Systems Based on Identification Information. In Advances in Cryptology, Crypto 1987, pp. 194-202. LNCS Vol. 293/1988.
32. E. Okamoto and K. Tanaka. Key Distribution System Based on Identification Information. IEEE Journal on Selected Areas in Communications, 7(4):481-485, May 1989.
33. E. Rescorla. SSL and TLS: Designing and Building Secure Systems Addison-Wesely. 2001. overhead.

34. R. Sakai, K. Ohgishi and M. Kasahara. Cryptosystems based on pairing. In Symposium on Cryptography and Information Security, Okinawa, Japan, 2000.
35. A. Shamir. On the Generation of Cryptographically Strong Pseudorandom Sequences. *ACM Trans. Comput. Syst.*, 1(1):38–44, 1983.
36. Adi Shamir: Identity-Based Cryptosystems and Signature Schemes. CRYPTO 1984: 47-53
37. Christian Schridde, Matthew Smith and Bernd Freisleben, An Identity-Based Key Agreement Protocol for the Network Layer. In SCN'2008.
38. Shmuely, Z., "Composite Diffie-Hellman Public-Key Generating Systems are Hard to Break", Technical Report 356, CS Dept, Technion, Israel, 1985.
39. V. Shoup, "Lower Bounds for Discrete Logarithms and Related Problems", Eurocrypt'97, LNCS 1233, pp. 256-266.
40. Shoup, V., On formal models for secure key exchange (version 4), November 15, 1999 . `http://www.shoup.net/papers/`
41. Smetters, D.K., Durfee, G.: Domain-based Administration of Identity-Based Cryptosystems for Secure E-Mail and IPSEC. In: SSYM 2003: Proceedings of the 12th Conference on USENIX Security Symposium, Berkeley, CA, USA, p. 15. USENIX Association (2003)
42. Y. Wang. Efficient Identity-Based and Authenticated Key Agreement Protocol. Cryptology ePrint Archive, Report 2005/108, 2005. http://eprint.iacr.org/2005/108/.

A Security Model for Key Agreement Protocols

The security analysis in this work is based on the Canetti-Krawczyk security model for key-agreement protocols [6, 7]. Here we present an abridged description of the CK model adopted from [25] where the model is specialized to the case of 2-message protocols and is extended to account for KCI resistance and weak perfect forward secrecy (properties not included in the basic CK model). A slight adaptation of the model for the identity-based setting is also described.

Throughout the paper, whenever we refer to a key agreement protocol as secure we mean secure in the sense of the Canetti-Krawczyk model as described here. The reader familiar with this model can skip this section. For full details consult [6].

A key-agreement (KA) protocol is run in a network of interconnected parties, which can be activated to run an instance of the protocol called a session. Within a session a party can be activated to initiate the session (the initiator) or to respond (the responder) to an incoming message. As a result of these activations, and according to the specification of the protocol, the party creates and maintains a session state, generates outgoing messages, and eventually completes the session by outputting a session key and erasing the session state. A session may also be aborted without generating a session key. A KA session is associated with its holder (the party at which the session exists), a peer (the party with which the session key is intended to be established), and a session identifier. We will simplify the presentation here by making two assumptions (consistent with the model of [6]): (i) the activation of a session at a party always specifies the name of the intended peer; and (ii) the session identifier is a quadruple $(Alice, Bob, Out, In)$ where Alice is the identity of the holder of the session, Bob the identity of its peer, Out and In are the outgoing and incoming messages (resp) at the holder. In particular, in the case of the mOT protocol, a session identifier will have the form $(Alice, Bob, \alpha, \beta)$. If Alice holds a session $(Alice, Bob, \alpha, \beta)$ and Bob holds a session $(Bob, Alice, \beta, \alpha)$ (for same values α, β) then we say that the two sessions are matching. Matching sessions play a fundamental role in the definition of security.

Note on notation: For ease of presentation we use the names Alice and Bob as identities of parties in the system. When convenient, however, we will denote these identities by id_A and id_B, respectively. Similarly, we will use S_A and S_B to denote the private keys of these parties.

In the Identity Based (IB) setting, there is an additional entity called the the Key Generation Center (KGC) that is responsible for generating public and private master parameters from which

users' private keys are derived. Each user A (or Alice) has a unique identity id_A and it receives from the KGC a private key S_A. Parties running a key agreement in this setting need to know the identity of their peers and the public parameters of the TA. There is no need for public key certificates as the identities themselves (or a publicly computable value derived from the identity, for example via hashing) function as public keys.

Attacker Model. The attacker is modeled to capture realistic attack capabilities in open networks, including the control of communication links and the access to some of the secret information used or generated in the protocol. The basic principle is that since security breaches happen in practice (e.g., via break-ins, mishandling of information, insider attacks, cryptanalysis), a well-designed protocol needs to *confine* the damage of such breaches to a minimum.

The attacker is an active "man-in-the-middle" adversary with full control of the communication links between parties. It can intercept and modify messages sent over these links, delay or prevent their delivery, inject its own messages, interleave messages from different sessions, etc. (Formally, it is the attacker to whom parties hand their outgoing messages for delivery.) The attacker also schedules all session activations. In addition, in order to model potential disclosure of secret information, the attacker is allowed access to secret information via session exposure attacks (a.k.a. known-key attacks) of three types: state-reveal queries, session-key queries, and party corruption. A state-reveal query is directed at a single session while still incomplete (i.e., before outputting the session key) and its result is that the attacker learns the session state for that particular session (which may include session-specific information but not long-term secrets). A session-key query can be performed against an individual session after completion and the result is that the attacker learns the corresponding session-key. Finally, party corruption means that the attacker learns *all* information in the memory of that party (including the long-term private key of the party as well all session states and session keys stored at the party); in addition, from the moment a party is corrupted all its actions may be controlled by the attacker. Indeed, note that the knowledge of the private key allows the attacker to impersonate the party at will.

Sessions against which any one of the above attacks is performed (including sessions compromised via party corruption) are called exposed. In addition, a session is also called exposed if its matching session has been exposed (since matching sessions output the same session key, the compromise of one inevitably implies the compromise of the other).

The security of session keys is captured via the inability of the attacker to distinguish between the real session key of an unexposed session and a random value. More precisely, the attacker is allowed to choose a *test session* among the complete and unexposed sessions in the protocol. In this case, the attacker receives a value v which is chosen as follows: a random bit b is tossed, if $b = 0$ then v is the real value of the session key, otherwise v is a random value chosen under the same distribution of session-keys produced by the protocol but independent of the value of the real session key. After receiving v the attacker may continue with the regular actions against the protocol; at the end of its run it outputs a bit b'. The attacker succeeds in its distinguishing attack if (1) the test session is not exposed, and (2) the probability that $b = b'$ is significantly larger than $1/2$.

Definition 1. [6] *A polynomial-time attacker with the above capabilities is called a* KA-attacker. *A key-agreement protocol π is called* secure *if for all KA-attackers running against π it holds:*

1. *If two uncorrupted parties complete matching sessions in a run of protocol π under the attacker then except for a negligible probability, the session key output in these sessions is the same.*
2. *The attacker succeeds (in its test-session distinguishing attack) with probability not more that 1/2 plus a negligible fraction.*

This definition captures the essential security properties of a KA protocol; in particular, [6] show that a KA protocol that is secure in the above sense is sufficient for the most important application of KA protocols, namely, the establishment of secure communication channels between two parties.

Additional properties: PFS and KCI. A very important property of key-agreement protocols not captured in the above definition is forward secrecy namely, the assurance that once a session key is erased from its holders memory then the key cannot be learned by the attacker even if the parties are subsequently corrupted. Formally, this is captured in [6] via the notion of session-key expiration (which represents the erasure of a session key from memory). A key-agreement protocol is said to be secure with perfect forward secrecy if the above definition holds even when the attacker is allowed to corrupt a peer to the test session *after the test-session key expired* at that peer. Note that this definition of PFS assumes an active attacker (as all the rest of the security model), that is, the forward security of the session key holds even for sessions (established between honest parties) where the attacker had an active role such as providing part of the messages in the session. A weaker version of this property, referred to as *weak PFS* (and formalized in [25]), only guarantees forward security for sessions established without active intervention of the attacker.

Yet another security property desirable for key-agreement protocols is their resistance to key-compromise impersonation (KCI) attacks. In this setting the attacker compromises Alice and learns her secret key. This knowledge clearly enables the attacker to impersonate Alice to others. Yet if this knowledge allows the attacker to impersonate other, uncorrupted parties, to Alice then we say that this is a KCI attack. We refer the reader to [25], for a formal treatment of this notion. We show our protocols to be resistant to this attack.

Extension to the identity-based setting. In our model each party has a unique identity that can be chosen by the attacker (even for honest parties). The only restriction is that different parties are given different identities. This models the fact that while we do not put any restrictions on the form of an identity we do assume that the KGC checks the validity of registered identities and that valid identities are all different. Very importantly, in the IB setting we allow the attacker to corrupt the KGC and find its master secrets. The meaning of this action is equivalent to corrupting *all parties*. However, even in this case it is important to provide security for past session keys. In other words, we are interested to preserve perfect forward secrecy (PFS), described above, even in the case of KGC corruption. This crucial security property is often referred to in the IB literature as KGC-FS (Key Generation Center – Forward Security). One of our important contributions is to show that the mOT protocol does enjoy KGC-FS.

Note on the mOT protocol and state-reveal queries. The mOT protocol analyzed in this paper is not resistant to the disclosure of the ephemeral Diffie-Hellman values or the unhashed session key. Therefore, when defining the protocol we will explicitly specify that these values be protected (e.g., via local encryption) with the same security as the long-term private key. (This is similar to the level of protection required by the DSA signature for its ephemeral exponent k.) At a formal level, this means that the attacker does not benefit from state-reveal queries; it can only learn the ephemeral values via full party corruption (as is the case with other key-agreement security models such as the Bellare-Rogaway model).