One-Pass HMQV and Asymmetric Key-Wrapping

Shai Halevi Hugo Krawczyk
IBM Research

December 22, 2010

Abstract

Consider the task of asymmetric key-wrapping, where a key-management server encrypts a cryptographic key under the public key of a client. When used in storage and access-control systems, it is often the case that the server has no knowledge about the client (beyond its public key) and no means of coordinating with it. For example, a wrapped key used to encrypt a backup tape may be needed many years after wrapping, when the server is no longer available, key-wrapping standards have changed, and even the security requirements of the client might have changed. Hence we need a flexible mechanism that seamlessly supports different options depending on what the original server was using and the current standards and requirements.

We show that one-pass HMQV (which we call HOMQV) is a perfect fit for this type of applications in terms of security, efficiency and flexibility. It offers server authentication if the server has its own public key, and degenerates down to the standardized DHIES encryption scheme if the server does not have a public key. The performance difference between the unauthenticated DHIES and the authenticated HOMQV is very minimal (essentially for free for the server and only 1/2 exponentiation for the client). We provide a formal analysis of the protocol's security showing many desirable properties such as sender's forward-secrecy and resilience to compromise of ephemeral data. When adding a DEM part (as needed for key-wrapping) it yields a secure signcryption scheme (equivalently a UC-secure messaging protocol).

The combination of security, flexibility, and efficiency, makes HOMQV a very desirable protocol for asymmetric key wrapping, one that we believe should be incorporated into implementations and standards.

1 Introduction

Key management is an essential component of secure systems, it is used in many systems both for confidentiality and for enforcing access-control policies. In this work we deal with settings where we have a key-server that needs to securely send a symmetric key to a client. The symmetric key could be a freshly generated one (to be used later by Alice) or it could be a pre-set key (e.g., to let Alice decrypt a previously encrypted file). We call the server Bob (or the sender) and call the client Alice (or the receiver). We specifically focus on settings where the protocol must be one-way, with the server sending a single message to the client with all the required information.

For example, consider a typical tape-encryption setting, where backup tapes are encrypted and then stored for future need by a potential client Alice who may need them one day. To enable decryption, Bob "wraps" the encryption key under some public key and stores the wrapped key to the tape itself. Note that Alice is off-line (or perhaps does not even exist) when the encryption key is wrapped, so no interaction can take place. Later, when Alice comes to need the backup tape, she asks her key-management module for the private key corresponding to the public key used by Bob, and then she can unwrap the decryption key and decrypt the backup tape. Such one-way communication can be viewed as *one-pass key-exchange protocols* with implicit client authentication (and in some cases also server authentication). In the case that a pre-set key is transmitted (as in the tape encryption example), the operation is referred to as *key wrapping*.

Key-wrapping (and key-management in general) is a topic of intensive activity in the industry, with many competing services and many standardization efforts (e.g., PKCS #11, FIPS SP 800-57 and 800-130, IETF KeyProv, OASIS KMIP, IEEE 1619.3, IEEE P1363, etc.) Symmetric key-wrapping was addressed in the work of Rogaway and Shrimpton [15] and later Gennaro and Halevi [8], with a focus on using deterministic encryption for this purpose. In this work, however, we focus on asymmetric key-wrapping, where Bob uses Alice's public key to wrap the symmetric key. In this context the added complexity of using randomization is insignificant in comparison to the public-key operations that are needed. Hence we just use standard encryption, and view key-wrapping as a target application rather than a separate security goal.

Many real-world implementations of asymmetric key wrapping are based on RSA, but the increase in the use of elliptic-curve cryptosystems suggest that they will be useful for key wrapping as well. The leading mechanisms in this respect is the "elliptic-curve integrated encryption scheme" (ECIES) [11], which is based on the "Diffie-Hellman integrated encryption scheme" (DHIES) encryption scheme of Abdalla et al. [1]. DHIES is an Elgamal-based encryption scheme, proven CCA secure under the "hashed Diffie-Hellman" assumption. (Alternatively, under the Gap Diffie-Hellman assumption in the random-oracle model.)

On a high level, in DHIES Alice has a secret exponent a, and the corresponding public key is the group element $A = g^a$ (where g is a generator in a prime-order group). To encrypt a message M, Bob chooses a random exponent y and computes $Y = g^y$, then computes the Diffie-Hellman value $\sigma = A^y$ and uses $K = H(\sigma, Y)$ as a key in a symmetric-key authenticated-encryption scheme to encrypt M. In more details, Bob derives from K two symmetric keys K_a, K_e, computes the ciphertext $C = \textsc{Enc}_{K_e}(M)$ where $\textsc{Enc}$ is some standard CPA-secure symmetric encryption scheme (e.g., AES-CBC), and also computes an authentication tag $T = \mathrm{MAC}_{K_a}(C)$ where MAC is some standard authentication code (e.g., HMAC-SHA1). Bob sends (Y, C, T) to Alice, who recovers the Diffie-Hellman value via $\sigma = Y^a$, then computes $K = H(\sigma, Y)$, re-derives K_a and K_e, verifies the authentication tag T and decrypts the ciphertext C to recover M. Note that DHIES is an instance of the KEM/DEM paradigm [16], where Y is the "key-encapsulation module" (KEM) part of the ciphertext and (C, T) is the "data-encapsulation module" (DEM) part.

When used for key-wrapping (i.e., when M is a cryptographic key), this encryption scheme can also be viewed as a key-exchange protocol where only the client is implicitly authenticated. However, there are applications where the server too should be authenticated. For example, consider the tape-backup application from above where Alice is a third party that provides backup/restore services. When Alice gets a tape with a wrapped key on it, she wants to consult the policy of the original server Bob to know if decryption is permitted. So in particular Alice needs to be able to authenticate the source of the wrapped key.

A natural solution is for Bob to sign the ciphertext (e.g. with ECDSA signature if we want an elliptic-curve scheme), but this solution has some drawbacks. For one thing, it adds non-trivial complexity. We need to implement the signature scheme and use additional bandwidth to communicate the signatures. Also, in some cases this does not even provide adequate authentication, since it allows an adversary Charlie to strip the original Bob signature from the wrapped key and replace it with a signature by Charlie. (This may enable Charlie to later ask Alice to unwrap the key and decrypt the tape for him.)

Instead, one would like a solution that (a) ensures that the identity of Bob cannot be stripped from the key, and (b) remains as close as possible to the base DHIES scheme. A good candidate to achieve this goal has been suggested in [12], namely the one-pass HMQV protocol. However, while a main appeal of HMQV is its provable security, the one-pass variant of HMQV has not been proven. Here we give a full specification of a one-pass HMQV protocol, with a full proof of security. We call this protocol HOMQV (for **H**ashed-**O**ne-pass-**MQV**, pronounced "Home-Queue-Vee"). We note that HOMQV is different than the one-pass HMQV protocol in [12, Sec. 9] in that HOMQV hashes the session identifier while deriving the session key (as suggested in [13]).

Roughly speaking, the only difference between HOMQV and (the KEM part of) DHIES is the way the Diffie-Hellman value σ is computed. Whereas in DHIES we set $\sigma = A^y = Y^a$, here Bob also has public and secret keys ($B = g^b$) so we use them in the computation. Roughly, we compute a half-size exponent $e = \bar{H}(Y)$ and then set $\sigma = (YB^e)^a = A^{y+be}$ and $K = H(\sigma, Y)$. (In the actual protocol we also add the identities of Alice and Bob inside the hashing and use cofactor exponentiation, see Table 1 for an overview and Section 3 for a precise definition of HOMQV.)

We also show that slight variations of the same base protocol can handle a large variety of scenarios in the key transmission and key wrapping settings. For example, a server Bob that does not have a public key can just use the dummy public key $B = 1$, and then the protocol reverts to (the KEM part of) the underlying DHIES. Other variants of the protocol offer increased security at a minimal cost in computation and communication. In all cases the schemes provide forward secrecy for the server, namely the compromise of the sender's private key (if any) does not expose past transmissions.[1] This property has particular importance for our setting, since in key-management and storage applications some of the keys may have a very long lifetime. In some variants this property is achieved in a weak sense, namely, as long as the attacker was passive during these transmissions, but we also show inexpensive variants that provide full sender forward secrecy, even against active attacks.

Remarkably, all the variants except the (degenerate) case of DHIES enjoy the security property that leakage of the ephemeral Diffie-Hellman exponent y does not compromise security; the function of this exponent is to provide forward-secrecy capabilities. We call this property *y-security.* The significance of this property is that it allows pre-computing (y, g^y) pairs, even if they are stored in

[1] On the other hand, it is easy to see that one-pass protocols inherently cannot offer PFS for the receiver.

KEM modes	$\hat{B}$ sends to $\hat{A}$	Key-derivation	impl.auth.	y-secure	Sender FS	#exp (s/r)
DHIES KEM	Y	$SK = K$	$\hat{A}$ only	No	No	2/1
HOMQV	Y	$SK = K$	$\hat{B}, \hat{A}$	Yes	weak	2/1.5
HOMQV + key-conf.	Y, $\mathrm{MAC}_{K_a}(1)$	$SK = PRF_K(0)$ $K_a = PRF_K(1)$	$\hat{B}, \hat{A}$	Yes	full	2/1.5

ENC. modes	$\hat{B}$ sends to $\hat{A}$	Key-derivation	DEM	security	#exp (s/r)
DHIES	Y, C, T	$K_a = PRF_K(1)$ $K_e = PRF_K(2)$	$C = \mathrm{E{\scriptsize NC}}_{K_e}(M)$ $T = \mathrm{MAC}_{K_a}(C)$	CCA	2/1
HOMQV + DEM	Y, C, T	$K_a = PRF_K(1)$ $K_e = PRF_K(2)$	$C = \mathrm{E{\scriptsize NC}}_{K_e}(M)$ $T = \mathrm{MAC}_{K_a}(C)$	signcryption	2/1.5

Table 1: One-pass modes. In all $Y = g^y$, $\sigma = (YB^e)^a = A^{y+be}$, $e = \bar{H}(\hat{A}, Y)$, $K = H(\sigma, \hat{B}, \hat{A}, Y)$, $\hat{B}, \hat{A}$ represent the identities of sender and receiver. In DHIES we use $B = 1, b = 0$.
Last column counts number of exponentiations per sender/receiver.

less secure media.

We have three variants for the basic setting of a one-pass KEM-only protocol: From the "degenerate" DHIES (that offers only CCA security), via plain HOMQV (that offers server-authentication and server-PFS against passive attacks), to a variant with key-confirmation that adds server-PFS also against active attacks. When adding the DEM part, we get two variants of encryption (or key-wrapping): Without server authentication we only get the CCA-secure DHIES encryption, and with server authentication we get a secure signcryption scheme (equivalently, UC-secure replayable message transmission). These variants (with a slight simplification) are summarized in Table 1, detailed description are found in Sections 3, 4, and 5.4.1.

We note that the precise connection between key-exchange and signcryption was established in the work of Gorantla et al. [10], and a comprehensive theory of KEM/DEM for signcryptions was developed by Dent [5, 7, 6]. See Section 4.

As all these schemes are one-pass protocols, they are all inherently open to replay attacks. This can be addressed by having the sender and receiver maintain synchronized state via counters or timestamps, or having the receiver store all past communication (or a combination of both). Alternatively, in Section 6 we show that HOMQV extends smoothly to the interactive setting, where we can prevent replay and offer full PFS for both sides.

In this paper we analyze all these variants and prove their security in the Canetti-Krawczyk model of key exchange [3] (specialized to the case of one-pass protocols). We prove both "basic security" of these variants as well as the additional properties of forward security and y-security (i.e., resilience to leakage of the ephemeral Diffie-Hellman exponents). Importantly, using these additional features we can appeal to the connections proven in [10] to conclude that HOMQV+DEM is a secure signcryption scheme (which is also a secure UC realization of the functionality of replayable message transmission, see [9]).

We highlight some advantages of HOMQV that make it attractive for applications such as key-wrapping for storage. One advantage is its very minimal overhead as compared to DHIES. For the sender Bob there is essentially no added cost, the only change is that it computes the Diffie-Hellman value as $\sigma = A^{y+be}$ rather than $\sigma = A^y$. This only entails an additional hashing (to compute e) and an addition and multiplication modulo the group order, but no additional exponentiations. For the receiver there is an additional 1/2 exponentiation, since it now computes the Diffie-Hellman value as $\sigma = (YB^e)^a$ rather than $\sigma = Y^a$. We mention that as described in Table 1, HOMQV and DHIES require a check that the elements belong to a prime-order sub-group (which can be as costly as one full exponentiation). In Section 3 we show how this check can often be avoided.

Another important advantage is the versatility of this protocol and the fact that it smoothly degenerates back to DHIES. This is important in particular to storage applications where keys that were wrapped long ago may be needed again in the far future and the standard that was used for wrapping underwent several changes in between. For example, imagine a client module implementing HOMQV that tries to access a key that was wrapped ten years ago with DHIES. The fact that HOMQV degenerates naturally to DHIES with $B = 1$ means that there is very little complexity (if at all) that needs to be added to the client to be able to handle such old wrapped keys. Similarly, the fact that HOMQV extends smoothly up to HMQV means that very minor implementation changes are needed to take advantage of the interactive setting, when available.

2 Security Model for One-Pass Key-Exchange Protocols

We specialize the Canetti-Krawczyk (CK) security model [3] to one-pass key-exchange protocols. A key-exchange (KE) protocol is run in a network of connected parties, where each party can be activated to run an instance of the protocol called a session. During the session, a party creates and maintains a session state, may send and receive 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 KE session is associated with its holder or owner (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. In one-pass protocols a peer to a session is either a sender or a receiver.

For simplicity we assume below that a session is always activated with the name of the intended peer (this is called "pre-specified peer" in [4]) and the session identifier is a triple $(\hat{B}, \hat{A}, Y)$ where $\hat{B}$ is the sender identity, $\hat{A}$ the receiver identity, and Y is the message sent in the protocol. (For HOMQV, Y is the Diffie-Hellman public value.) Two sessions with the same identifier are called matching. Matching sessions play a fundamental role in the definition of security.

Each party owns a long-term pair of private and public keys, and other parties can verify the binding between an identity and a public key (e.g., using a CA or manually, the exact mechanism is outside the scope of this paper). A corrupted party can choose at any point to "register" any public key of its choice, including public keys equal or related to keys of other parties in the system (and there is no requirement that it knows the corresponding private key). In this paper, a public key will always be a group element, and the private key its secret exponent.

Notations and identities. A "hat" on top of a capital letter denotes an identity; without the hat the letter denotes the public key of that party, and the same letter in lower case denotes a private key. For example, Alice has identity $\hat{A}$ and a public key $A = g^a$ with a as the private key.[2]

Attacker Model. The attacker's capabilities as modeled below are intended to capture many realistic attacks, including key-compromise. The attacker, denoted $\mathcal{M}$, is an active "Man-in-the-Middle" adversary with full control of the communication links between parties. $\mathcal{M}$ can read, modify, inject, delete, or delay messages at will. (Formally, it is $\mathcal{M}$ to whom parties hand their outgoing messages for delivery.) $\mathcal{M}$ also schedules all session activations and session-message delivery. In addition, in order to model potential disclosure of secret information, the attacker is

[2] This notation assumes that there is *a unique public key* associated with each identity. In the real world, where a party may have more than one public key, the symbol $\hat{A}$ is assumed to include an indication of a unique public key.

allowed access to secret information via session exposure 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 (such as the secret exponent of an ephemeral public DH value). A session-key query can be performed against a single session after completion and the result is that the attacker learns the corresponding session-key (this models leakage on the session key either via usage of the key by applications, cryptanalysis, break-ins, known-key attacks, etc.). 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 are 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 exposed if the matching session has been exposed (since matching sessions must output the same session key, the compromise of one inevitably implies the compromise of the other).

Note on state-reveal queries. "State-reveal queries" in the CK model is meant to capture the distinction between secrets whose exposure will derail security "forever" (such as long-term keys, or the ephemeral exponent in a DSA signature) and secrets whose exposure only compromises the current session. Data that can be accessed via such state-reveal queries is thought of as "less secret", its exposure only compromises the current session (and thus is can perhaps be stored in less secure memory in a real implementation). Data not accessible via state-reveal is assumed to get the same protection of long-term secrets, such data is revealed to the adversary only upon full compromise of a party.

Basic security. The security of session keys generated in unexposed sessions is captured via the inability of the attacker $\mathcal{M}$ to distinguish the session key of a test session (chosen by $\mathcal{M}$) from a random value. When $\mathcal{M}$ chooses the test session, a random bit b is drawn and $\mathcal{M}$ gets either the real value of the session key (if $b = 1$), or an unrelated random value (if $b = 0$). The attacker can continue with the regular actions against the protocol also after the test session; at the end of its run $\mathcal{M}$ outputs a guess b' for the value of b. The attacker succeeds in its distinguishing attack if (1) the test session is not exposed, and (2) it guessed correctly, $b = b'$. The protocol satisfies the basic notion of security if feasible attackers cannot succeed with probability significantly better than 1/2.

Definition 1 ([3]) *A polynomial-time attacker with the above capabilities is called a* KE-attacker*. A key-exchange protocol π is called* secure *if for all KE-attackers $\mathcal{M}$ running against π it holds:*

1. *If two uncorrupted parties complete matching sessions in a run of protocol π under attacker $\mathcal{M}$ then they output the same key (except for a negligible probability).*

2. *$\mathcal{M}$ succeeds (in its test-session distinguishing attack) with probability not more that 1/2 plus a negligible fraction.*

Sender's forward secrecy. An important property that is not captured by basic security is perfect forward secrecy (PFS) [14], namely the assurance that a session key which is erased from memory cannot be learned by the attacker even if the long-term key of that party is later exposed. This is captured formally in [3] via the notion of session-key expiration (which represents the erasure of a session key from memory). A key-exchange protocol is secure with PFS if Definition 1 holds even when the attacker is allowed to corrupt a peer to the test session *after the test-session key expired* at that peer. In the case of one-pass protocols, forward secrecy cannot be provided in general (since exposing the receiver's private key clearly lets the attacker decrypt all incoming traffic), but the sender can still enjoy forward secrecy. As we will see, the basic HOMQV protocol provides sender's forward secrecy against passive attackers, and adding key-confirmation provides forward secrecy also against active attackers.

Replay attacks. A one-pass key-exchange protocol where parties do not maintain evolving state between sessions is always open to replay attacks, where the attacker forces the establishment of the same session at a receiver by replaying old incoming messages to that party. The model deals with this issue by defining all instances of such replayed sessions to be matching. This means that all these sessions have the same session key and also that for the attacker to be considered successful in attacking a session it should have not exposed any one of the session copies.

3 The Basic HOMQV Protocol

On groups and supergroups. The protocol uses a cyclic group G of prime order q generated by a given generator g. There is no particular requirement from the cyclic group $G = \langle g \rangle$ except for its prime order. However, in cases where G is a subgroup of another group G' and testing membership in G' is easier than testing membership in G, we will exploit this property in the protocol. For this we define f as the "cofactor" $f = |G'|/|G|$. For example, if g is an element of Z_p^* of prime order q, then G' is Z_p^* and $f = (p-1)/q$. If g is an element in an elliptic curve group E then $f = |E|/q$. We always assume that f, q are co-primes.

Using the cofactor, we describe a protocol where one only needs to verify that the various elements are in G' (an efficient test in the above two examples). Note that it can be the case that $f = 1$. The property of the cofactor f that we use is that $X^f \in G$ for any $X \in G'$.

Hash functions. The protocol uses two hash functions (which are viewed as two independent random oracles): one, denoted $\bar{H}$, hashes strings into $\{0, 1, \dots, \sqrt{q}\}$ and is used to compute exponents of size $|q|/2$, the other, denoted H, hashes into $\{0,1\}^k$ where k is the length of the output key K.

Protocol HOMQV. Let $\hat{A}, \hat{B}$ be two *different* parties with keys $A = g^a$, $B = g^b$, respectively. To exchange a key with $\hat{A}$, sender $\hat{B}$ checks that $A \in G'$ (if not it aborts), then chooses random $y \in_{\mathrm{R}} Z_q$, and sends $Y = g^y$ to $\hat{A}$. $\hat{B}$ computes the key as $H(\sigma, \hat{B}, \hat{A}, Y)$ where $\sigma = A^{f \cdot (y+eb)}$ and $e = \bar{H}(Y, \hat{A})$. On incoming value Y and peer's identity $\hat{B}$, $\hat{A}$ checks that Y and $\hat{B}$'s public key B are in G' (if not, it aborts) and then computes the session key as $H(\sigma', \hat{B}, \hat{A}, Y)$ where $\sigma' = (YB^e)^{f \cdot a}$. Both parties compute the same session key since $\sigma = \sigma'$, and the session-id is the triple $(\hat{B}, \hat{A}, Y)$.

Performance. For the sender, the protocol requires just two exponentiations and a membership test in G' (the latter has negligible cost for the typical Z_p^* and elliptic curve cases). The exponentiations are for computing $Y = g^y$ and for computing $\sigma = A^{f \cdot (y+eb)}$ (In the latter case we first set $t = f \cdot (y + eb) \bmod \mathsf{ord}(G')$ and then σ is set to A^t. If $\mathsf{ord}(G')$ is not much larger than q — as in the case of elliptic curves — then the cofactor exponentiation is essentially for free.) The group element Y can be computed offline, even before knowing the identity of the peer.

The cost for the receiver $\hat{A}$ is 1.5 on-line exponentiations: a half exponentiation when computing (YB^e), and a full exponentiation for raising it to the power fa. (As before, computing $t' = fa \bmod \mathsf{ord}(G')$ is essentially for free when $\mathsf{ord}(G') \approx\ q$.) Comparing it with the cost of the unauthenticated DHIES, we see that *authentication is for free for the sender, and only costs 1/2 exponentiation for the receiver!*

Note, in particular, that there are no hidden costs in the protocol such as the need to test group elements for membership in the prime order subgroup G, only inexpensive G'-membership tests are needed. Also in contrast to some other protocols, HOMQV does not need the CA to checks that parties know the secret key for the public key that they want to register, thus avoiding complex proof-of-possession protocols.

A variant without the cofactor. In cases where $\mathsf{ord}(G') \gg q$ (as typically happens when working over Z_p^* with $q|p-1$), using the cofactor as above result in having to compute long multiplications in G', which could be much more expensive than exponentiations in the subgroup G. In these cases it may be better to use a variant without cofactor, and instead directly verify that the different elements are in the subgroup G: The sender $\hat{B}$ must verify that the receiver's public key A is in the subgroup G while the receiver $\hat{A}$ needs to test $YB^e \in G$ (there is no need to check Y and B separately). We also remark that verification of the receiver's public key A by the sender is only needed to get y-security; if we assume that the y value is never exposed then one can prove security even without that verification step, see more discussion in the appendix.

3.1 Notes on Security

Session-state exposure. As noted in Section 2, protocol designers should provide guidance to implementations regarding the consequences of exposure of different secret elements in the protocol. For the HOMQV protocol, one can readily see that disclosure of both the Diffie-Hellman value σ as well as of the exponent y from the same session leads to the exposure of the value $A^b = B^a$, which suffices for impersonating $\hat{B}$ to $\hat{A}$ and vice versa. So clearly, these two pieces of session-specific data should not be stored together in less secure memory.

How about disclosing any one of these values but not the other? For reasons similar to the above, if one discloses σ at the receiver then an attacker can do the following: chooses y, sends $Y = g^y$ to $\hat{A}$ purporting to be $\hat{B}$ (an honest player), then it finds σ at $\hat{A}$ and derives from it A^b which as said allows to impersonate $\hat{A}$ and $\hat{B}$ to each other. As for the secret exponent y, we show in Section 5.3 that its disclosure does not even compromise the session in which it is used. To have a negative effect, the attacker needs to find either the σ value of that session or the private key of the party that generated this exponent. This is an important property of HOMQV, since it means that the pair $(y, Y = g^y)$ can be generated in an off-line phase (even before knowing the identity of $\hat{A}$) and stored for later use. An exposure of such pair does not cause any real damage as long as the long-term key is not disclosed.

Replay. Being a one-pass protocol, HOMQV is obviously open to replay. This can be prevented by including synchronized timestamps (or a shared increasing counter). The timestamp or counter becomes part of the session-id together with $(\hat{B}, \hat{A}, Y)$ and hence included under H when computing the session key K. Another option is to use an interactive protocol in which $\hat{A}$ sends to $\hat{B}$ a nonce, which is then included in the computation of e. Note that the resulting interactive protocol is weaker than HMQV (e.g., it does not provide receiver PFS), but it is cheaper and prevents replay. See Section 6.

On Forward Secrecy. The protocol HOMQV as above provides forward security for the sender, namely, the guarantee that the disclosure of the private key of the sender $\hat{B}$ does not expose past sessions created by $\hat{B}$. However, an active attacker can choose $Y = g^y$ and send it to a receiver

$\hat{A}$, purporting to be $\hat{B}$. Later, if the attacker find $\hat{B}$'s private key then it can compute the session. Hence forward secrecy is only ensured against passive attackers. We can get full sender's forward secrecy by adding a key confirmation field in $\hat{B}$'s message, see Section 5.4.1. Receiver's forward secrecy is impossible in one-pass key exchange, since the receiver does not contribute any session-specific values.

KCI attacks. In a Key Compromise Impersonation attack (KCI), knowing the private key of a party $\hat{A}$ allows the attacker to impersonate other parties to $\hat{A}$. This is obviously possible in HOMQV.

4 Using HOMQV for Encryption and Key-Wrapping

On its own, HOMQV is used to establish a new secret key between the client and server. To get encryption or key-wrapping we need to add a "data encapsulation module" (DEM), where the new key is used in a symmetric encryption mode in order to encrypt (or wrap) the transmitted message/key. Specifically, we use the DEM part of DHIES, where the new key from HOMQV is expanded into encryption and authentication keys, which are then used in an Encrypt-then-Authenticate mode to send the message.

Namely, after computing $K = H(\sigma, \hat{B}, \hat{A}, Y)$ we set $K_a = PRF_K(1)$ and $K_e = PRF_K(2)$ (where 1,2 can be replaced by any two different fixed messages that are publicly determined by the protocol flow). Then to encrypt a message M we compute $C = \textsc{Enc}_{K_e}(M)$ where $\textsc{Enc}$ is a CPA-secure symmetric encryption scheme and then $T = \mathrm{MAC}_{K_a}(C)$ where MAC is a secure message-authentication code. The DEM part (C, T) — which is an authenticated encryption of M under the shared key (K_e, K_a) — is then added to the HOMQV flow to make the composite ciphertext (Y, C, T).

Regarding security, if we use the "degenerate case" of HOMQV where the public key B is set to 1 then we get exactly the DHIES encryption scheme, which was proven CCA secure by Abdalla et al. [1]. When adding the DEM part to the HOMQV scheme from Section 3 (with server authentication) we get a signcryption scheme [17], whose security can be argued as follows:

Step 1: signcryption-KEM. Gorantla et al. proved in [10, Thm 3] that a one-pass key-exchange protocol Π which is secure in the Canetti-Krawczyk model and offers sender forward secrecy, is also a signcryption-KEM scheme secure in the sense of insider confidentiality and outsider unforgeability in the multi-user setting. See [10] for the definitions of these notions of security. Roughly speaking this means that CCA-security of the encryption part holds even against an attacker that knows the sender's private key, while unforgeability of the signature holds only against an attacker that does not know the receiver's private key.

Moreover, one can verify that the proof from [10] works even if the protocol Π offers sender forward secrecy only against passive attackers. (Roughly speaking, this is because this property is only used with respect to the challenge ciphertext in the KEM-CCA game, which is generated honestly according to the signcryption algorithm.) Since we prove in Theorem 4 and Lemma 6 that HOMQV is secure in the Canetti-Krawczyk model and it offers sender forward secrecy against passive attackers, it follows that it is also a signcryption-KEM scheme secure in the sense of insider confidentiality and outsider unforgeability.[3]

Step 2: adding the DEM. A signcryption-KEM secure as above can be transformed into regular signcryption (secure in the same sense) by adding an authenticated-encryption DEM. Such results were proven by Dent [5] in slightly different models, and the same proofs work for our case as well. Namely, we have:

Lemma 2 *The combination of a signcryption KEM with insider confidentiality and outsider unforgeability in the multi-user setting, together with an authenticated-encryption DEM, yields a signcryption scheme with insider confidentiality and outsider unforgeability in the multi-user setting.*

Proof:(sketch) The confidentiality part against insider attacks is identical to the case of standard hybrid encryption. The integrity part against outsider attacks follows roughly from these arguments: (i) An outside attacker cannot generate a valid KEM for any symmetric key except those that were included in one of the ciphertexts that the attacker obtained from its oracle, due to the KEM integrity against outside attackers; (ii) The KEM confidentiality implies that the keys included in the ciphertext from the oracle are indistinguishable from random; and (iii) The integrity-of-ciphertext property of the DEM implies that the attacker cannot generate new valid DEM ciphertexts under these pseudo-random keys.

It therefore follows that augmenting HOMQV with DEM as above gives a signcryption scheme secure in the sense of insider confidentiality and outsider unforgeability in the multi-user setting. These arguments can be made formal via a standard sequence-of-games proof. □

Gjøsteen and Kråkmo proved in [9] that a signcryption scheme secure as above can be used in conjunction with PKI to realize a secure messaging functionality in the UC framework. (The messaging functionality is similar to Canetti's secure-message-transmission functionality from [2], except that signcryption does not prevent replay.) We thus conclude that when used in conjunction with PKI, the protocol HOMQV with the DEM part can UC-realize secure messaging. This UC-secure-messaging, in turn, can be used to transport keys from server to client, providing as much security as can be obtained from a one-pass protocol.

Why use HOMQV for Key-Wrapping? The main advantage of HOMQV over the underlying DHIES is server authentication: We let the client verify that wrapped keys indeed arrive from the correct server without adding much complexity or making any changes in the data path. This eliminates the need to rely on out-of-band authentication mechanisms that are presumably needed when using unauthenticated schemes such as DHIES and its variants.

Another advantage is the "y-security" of HOMQV. Recall that HOMQV remains secure even if the ephemeral secret exponent y is exposed, as long as the long-term secret key of $\hat{B}$ is protected. On the other hand, DHIES is clearly broken in this case. This provides a significant line of defense in settings where exposure of y is a real concern, such as when pairs $(y, Y = g^y)$ are precomputed and stored in less secure memory.

At the same time, the fact that HOMQV degenerates back to DHIES by using server public key $B = 1$ means that a HOMQV client can be used to unwrap legacy DHIES-wrapped keys with very little added complexity (if at all). In this case of course we would have to fall back to out-of-band authentication, but otherwise the legacy support is nearly transparent.

[3] Gorantla et al. already suggested using one-pass HMQV to get a secure signcryption KEM, but did not prove security for the variant of one-pass HMQV that they used.

5 Security Analysis of HOMQV

5.1 XCR Signatures and Gap-DH

XCR signatures are challenge-response signature where only the challenger can verify the signature. Specifically, the signer $\hat{B}$ has a private key $b \in_{\mathrm{R}} Z_q$ and a public key $B = g^b$. On input a message m and "challenge" $X = g^x$, the signature of $\hat{B}$ on m and challenge X is a pair $(Y = g^y, \sigma)$ where y is chosen by $\hat{B}$ at random in Z_q and σ is defined as $X^{f\cdot(y+\bar{H}(Y,m)b)}$. The challenger, who knows x, verifies the signature by checking that Y is in G' and that $\sigma = (YB^{\bar{H}(Y,m)})^{f\cdot x}$.

Note: In the original XCR signatures from [12] the cofactor f is not used; yet it is immediate to see that XCR remains secure with the cofactor: simply consider the original XCR challenge to be X^f. The cofactor is introduced here since it is needed in the analysis of HOMQV.

Security of XCR signatures. XCR is called secure if it is secure under a chosen-message (and chosen-challenge) attack in the following game between a forger $\mathcal{F}$ and a signing oracle $\hat{B}$. The input to $\mathcal{F}$ is $\hat{B}$'s public key $B \in_{\mathrm{R}} G$ and a challenge $X_0 \in_{\mathrm{R}} G$. $\mathcal{F}$ provides queries (X, m) to $\hat{B}$ which responds with a valid XCR signature (Y, σ). After polynomially many adaptive queries the forger wins if it outputs a valid signature (Y_0, σ_0) on any message m_0 using the (input) challenge X_0. The only requirement is "strong" existential unforgeability, namely, that either $\hat{B}$ was not queried for a signature on message m_0 or, if it was queried on m_0 it output a signature different than the pair (Y_0, σ_0).

Theorem 3 ([12]) *Under the CDH assumption, XCR signatures are secure in the random oracle model.*

Notes. The proof in [12] shows that when the output of $\bar{H}$ is ℓ-bit long and we have a forger over order-q subgroup that makes m random-oracle queries to $\bar{H}(\cdot)$ and succeeds in forging with probability $\frac{m^2}{2q} + 2^{-\ell} + \varepsilon$ (for some $\varepsilon > 2^{-\ell}$), then one can construct a CDH solver with comparable running time and success probability $\Omega((\varepsilon - 2^{-\ell})^2)$. Hence the output of $\bar{H}$ can be made as short as the security parameter (which in our case we take to be half the bit-size of q).

Also, for the proof of HOMQV we actually need a weaker property, namely, unforgeability against a 0-message attack (i.e., the attacker is given B and X_0 and needs to come up with a valid signature (Y, σ) for any message of its choice but without ever seeing a signature produced by $\hat{B}$).

The Gap-DH Assumption. The security of XCR is proved under the Computational Diffie-Hellman (CDH) Assumption over G, namely, given $U, V \in_{\mathrm{R}} G = \langle g \rangle$ it is infeasible to compute $DH_g(U, V)$ (the Diffie-Hellman function, with generator g, applied to U and V). To prove HOMQV we need the stronger Gap-DH assumption. We say that a decision algorithm $\mathcal{O}$ is a *Decisional Diffie-Hellman (DDH) Oracle* for a group G and generator g if on input a triple (U, V, W), for $U, V \in G$, oracle $\mathcal{O}$ outputs 1 if and only if $W = DH_g(U, V)$. We say that G satisfies the Gap-Diffie-Hellman (GDH) assumption if no feasible algorithm exists to solve the CDH problem, even when the algorithm is provided with a DDH-oracle for G. (After proving Theorem 4 we show that the Gap-DH assumption can be relaxed to a weaker assumption, which is not only sufficient but also necessary.)

5.2 Proof of Basic Security

Theorem 4 *In the random oracle model and under the Gap-DH assumption, HOMQV is a secure one-pass key-exchange protocol as per Definition 1.*

Proof: We need to prove that (1) Any two matching sessions with honest peers output the same session key; and (2) A KE-attacker has negligible advantage in winning a test-session experiment. Since sessions are matching if and only if they have the same session id $(\hat{B}, \hat{A}, Y)$, condition (1) follows from the fact that these three values determine a unique value for σ and hence for the session key. The proof of (2) works by reduction to the security of XCR signatures, assuming a decisional Diffie-Hellman oracle as in the Gap-DH assumption. That is, given an assumed KE-attacker $\mathcal{M}$ against HOMQV and a decisional DH oracle we build a (0-message) forger $\mathcal{F}$ against XCR.

Let N be such that the adversary $\mathcal{M}$ has non-negligible advantage in attacking the KE-security of HOMQV when used with N honest parties. $\mathcal{F}$ gets as input a public key B and challenge X_0 (both random elements in G). It starts by simulating a run of HOMQV with N honest parties, and selects two different parties at random among these N honest parties which we denote by $\hat{A}$ and $\hat{B}$; this is $\mathcal{F}$'s guess for the receiver and sender, respectively, of the test session to be chosen by $\mathcal{M}$. Using its inputs B and X_0, $\mathcal{F}$ sets $\hat{A}$'s public key to $A = X_0$ and $\hat{B}$'s public key to B (note that $\mathcal{F}$ does not know the private keys corresponding to these public keys). For all other honest parties, $\mathcal{F}$ chooses their private and public keys. (Corrupted parties have their PKs chosen by the attacker $\mathcal{M}$ who also manages all their actions.) $\mathcal{M}$ controls all activations, traffic and delivery of messages. Invocations of the random oracle ($\bar{H}$ and H) are usually answered by $\mathcal{F}$ by choosing a random response from the appropriate range, except as described in the special cases below. (If the same query is made more then once, then it gets the same answer every time.) In any case in which a corruption query is issued by $\mathcal{M}$ against $\hat{A}$ or $\hat{B}$, or if $\mathcal{M}$ chooses a test session with peers other than $\hat{A}$ as receiver and $\hat{B}$ as sender, $\mathcal{F}$ aborts its run (in both cases $\mathcal{F}$ failed to guess correctly the peers to the test session).

Any query (activation) of an honest party other than $\hat{A}$ or $\hat{B}$ is answered by $\mathcal{F}$ just as in the protocol, which $\mathcal{F}$ can do since it knows the secret keys of all these honest parties. Similarly, corruption of honest parties other than $\hat{A}$ or $\hat{B}$ are answered by $\mathcal{F}$ using the secret keys that it knows. It is left to show how to simulate queries to $\hat{A}$ and $\hat{B}$.

Send-query (potentially followed later by a session-key reveal query). Assume that the sender is $\hat{B}$, the case where the sender is $\hat{A}$ is handled in exactly the symmetric way. Denote the session's receiver by $\hat{C}$ and its public key by C. If $\hat{C} = \hat{B}$ of if $C \notin G'$ then $\mathcal{F}$ aborts the session. Otherwise $\mathcal{F}$ chooses y at random and sets $Y = g^y$ as the outgoing value to be sent.

If the receiver $\hat{C}$ is an honest party other than $\hat{A}$ then $\mathcal{F}$ computes the session key using $\hat{C}$'s secret key (as done in the protocol), and uses that value to answer secret-key reveal query against this session of $\hat{B}$ (if any). Otherwise (i.e., $\hat{C}$ is either $\hat{A}$ or a corrupted party), $\mathcal{F}$ chooses at random $K \in_{\mathrm{R}} \{0,1\}^k$ and uses this K for future secret-key reveal queries against this session of $\hat{B}$. From this point on, $\mathcal{F}$ will use its DDH oracle to check for each new H-query of the form $(Q, \hat{B}, \hat{C}, Y)$ (for some Q), whether $Q = DH_g(YB^{\bar{H}(Y,\hat{C})}, C^f)$. If so, $\mathcal{F}$ responds to that random-oracle query with the value K.

Note that due to the hashing of the session id in the computation of a session key, the value K is independent of any other session key except for the possible matching session held in this case by a corrupted party or $\hat{A}$. Thus, F's responses to other session-key queries (or corruption queries) are independent of K except if $\hat{C} = \hat{A}$ in which case the matching session at $\hat{A}$ is set to K as well.[4]

Receive-query (potentially followed later by a session-key reveal query). Assume that the receiver is $\hat{A}$, the case where the receiver is $\hat{B}$ is handled in exactly the symmetric way. Denote the session's sender by $\hat{C}$ and its public key by C, and denote the incoming value in this session by Z.

If $\hat{C} = \hat{A}$ or if C or Z are not in G' then $\mathcal{F}$ aborts the session. If (C, Z) was used before to activate $\hat{A}$ as receiver (because replay is possible), $\mathcal{F}$ sets the session key to its previous value. If $\hat{C}$ is an honest party other than $\hat{B}$, and Z was previously generated by an instance of this honest $\hat{C}$ in the role of a sender (so $\mathcal{F}$ knows z s.t. $Z = g^z$), then $\mathcal{F}$ computes the session key using this z together with $\hat{C}$'s secret key as done in the protocol, and uses that session key to answer secret-key reveal query against this session of $\hat{A}$ (if any).

Otherwise, we know that $\hat{C}$ is either $\hat{B}$ or a corrupted party, or $\hat{C}$ is honest but Z was never generated by an instance of $\hat{C}$ in the role of a sender. Then for each H-query of the form $(Q, \hat{C}, \hat{A}, Z)$ (for some Q) ever issued by $\mathcal{M}$, $\mathcal{F}$ uses its DDH oracle to check whether $Q = DH_g((ZC^{\bar{H}(Z,\hat{A})})^f, A)$. If such Q is found then $\mathcal{F}$ sets the session key to $K = H(Q, \hat{C}, \hat{A}, Z)$. Else, if $\hat{C} = \hat{B}$ and Z was indeed sent by a simulated session of $\hat{B}$ then $\mathcal{F}$ sets the session key to be the same value K that it chose when processing that previous Send query at $\hat{B}$. Else $\mathcal{F}$ sets the session key K as a new random ℓ-bit string. Thereafter, each time a session key reveal query is performed at a session $(\hat{C}, \hat{A}, Z)$, $\mathcal{F}$ responds with K. (As said, there may be more than one because of replay.) From now on, $\mathcal{F}$ uses its DDH oracle to check for any H-query of the form $(Q, \hat{C}, \hat{A}, Z)$ (for any Q) whether $Q = DH_g((ZC^{\bar{H}(Z,\hat{A})})^f, A)$ and if so it answers that query with K.

Test session. In any case where the test session is chosen with peers other than $\hat{A}$ as receiver and $\hat{B}$ as sender, $\mathcal{F}$ aborts its run. When the test session is chosen with $\hat{A}$ as receiver and $\hat{B}$ as sender, $\mathcal{F}$ returns to $\mathcal{M}$ with probability 1/2 the same answer K that it would have returned in a session-key reveal query (say, against the session in $\hat{B}$), and with probability 1/2 $\mathcal{F}$ returns just an independent random value K'.

As long as $\mathcal{M}$ does not query $H(\sigma, \hat{B}, \hat{A}, Y)$ with the values σ, Y corresponding to the test session, it has only probability 1/2 to answer correctly. This is because no session $(\hat{B}, \hat{A}, Y)$ was compromised and all other sessions have keys that are independent of K (since they were generated via H calls with different inputs). Therefore both K and K' are random values independent of $\mathcal{M}$'s view. By assumption $\mathcal{M}$ succeeds with non-negligible advantage, thus implying that $\mathcal{M}$ computes the correct σ with non-negligible probability.

Finally, whenever $\mathcal{F}$ guesses correctly the peers to the test session and $\mathcal{M}$ successfully generates the σ value corresponding to the test session ($\mathcal{F}$ identifies it via the decisional oracle), $\mathcal{F}$ stops its run and outputs the pair (Y, σ) where Y is the test session's outgoing value and $\sigma = A^{y+be}$ with $e = \bar{H}(Y, \hat{A})$. But this pair (Y, σ) is a *valid* XCR signature of $\hat{B}$ on message $\hat{A}$ and challenge $A = X_0$; since $\mathcal{F}$ has never queried $\hat{B}$ for any signature, $\mathcal{F}$ wins the forgery game. Since this happens with non-negligible probability, we obtain a contradiction to the security of XCR signatures, thus proving the theorem. □

Note regarding the Gap-DH queries. A valid query to a decisional oracle is one where the two inputs are elements of G. In the above proof, this is indeed the case for the $Q \overset{?}{=} DH_g(YB^{\bar{H}(Y,\hat{C})}, C^f)$ query since Y, B and C^f are all in G (Y by $\mathcal{F}$'s choice, B as an input to $\mathcal{F}$, and C^f since $\mathcal{F}$ checks that $C \in G'$). The same is the case for the query $Q \overset{?}{=} DH_g((ZC^{\bar{H}(Z,\hat{A})})^f, A)$ since both Z and C

[4] Without hashing the session id, a UKS attack is possible where different, non-matching, sessions have the same key.

are checked to be in G' thus the value $(ZC^{\bar{H}(Z,\hat{A})})^f$ is necessarily in G. Note that if the cofactor f is not used in the protocol then the sender needs to check explicitly that the receiver's public key C is in G (i.e., an element of G' of order q) and the receiver needs to check that the value $ZC^{\bar{H}(Z,\hat{A})}$ is in G.

Gap-DH vs Strong-DH Assumptions. For simplicity, we have stated the theorem and proof using the Gap-DH assumption defined above. A weaker assumption, called the Strong DH assumption, states that computing $DH_g(A, B)$ for a given pair $A, B \in_{\mathrm{R}} G$ is infeasible even when given two *specific* decisional oracles $\mathcal{O}_A$ and $\mathcal{O}_B$ defined as follows: for all $C, D \in G$, $\mathcal{O}_A(C, D)$ returns true iff $DH_g(A, C) = D$ and $\mathcal{O}_B(C, D)$ returns true iff $D = DH_g(B, C)$. This weaker assumption is sufficient to prove the theorem. Indeed, the query $Q \overset{?}{=} DH_g(YB^{\bar{H}(Y,\hat{C})}, C^f)$ in the proof (the Send query case) is the same as $DL_g(Q) \overset{?}{=} (y + e \cdot DL_g(B)) \cdot DL_g(C^f)$. This is equivalent to $DL_g(Q/C^{fy}) \overset{?}{=} DL_g(B)) \cdot DL_g(C^f) \cdot e$, namely $Q/C^{fy} \overset{?}{=} DH_g(B, C^{fe})$. The latter test is just the query $\mathcal{O}_B(C^{fe}, Q/C^{fy})$, which $\mathcal{F}$ can make since it knows y and e. The other query to the DDH oracle in the proof (the Receive query case) is $Q \overset{?}{=} DH_g((ZC^{\bar{H}(Z,\hat{A})})^f, A)$, which can similarly be expressed in terms of $\mathcal{O}_A$.

It is interesting to note that the Strong-DH assumption is necessary for the proof since the protocol itself provides a decisional oracle with respect to the public key of any honest party. For example, if an attacker $\mathcal{M}$ is interested to check whether $DH_g(A, C) = D$ for given values C and D (where A is the public key of an uncorrupted party $\hat{A}$), then $\mathcal{M}$ proceeds as follows. It chooses a value $Z \in G$, sets $e = H(Z, \hat{A})$ and $M = (C/Z)^{1/e}$. Then it activates $\hat{A}$ as receiver with incoming value Z and incoming public key M (and sender's identity $\hat{M}$). The session key computed by $\hat{A}$ is $K = H((ZM^e)^a, \hat{M}, \hat{A}, Z) = H(C^a, \hat{M}, \hat{A}, Z)$ which equals $H(D, \hat{M}, \hat{A}, Z)$ iff $DH_g(A, C) = D$. By querying the session key K, $\mathcal{M}$ learns whether $DH_g(A, C) = D$. Finally, we note that the DDH oracle for the sender's public key is only needed to obtain y-security (i.e., security even in the case of a disclosed ephemeral value y); in all other cases, a DDH oracle with respect to the receiver's public key suffices. This can be shown by using the alternative proof in Lemma 8.

Beyond basic security. As discussed in Section 2, a major goal in the design of cryptographic protocols is to minimize the negative effects of secret data exposure. The proof of Theorem 4 shows this to be the case for HOMQV with respect to the disclosure of session keys (e.g., exposure of one session key has no effect on the security of other session keys). We now show security at a finer granularity: Below we show that HOMQV is strongly resilient to the disclosure of ephemeral DH exponents and also that it provides sender's forward secrecy in case of the disclosure of the private key of a party.

5.3 Resilience of HOMQV to Disclosure of Ephemeral Exponents

We prove that HOMQV has maximal resilience to the leakage of the ephemeral DH exponents y used by a sender to produce outgoing DH values $Y = g^y$ (we referred to this property as y-security). Indeed knowledge of these y's by an attacker (even ahead of their use) does not even compromise the security of the sessions where they are used, except of course if the attacker also learns the private key of the sender. The practical meaning is that it is as safe as possible to compute pairs $(y, Y = g^y)$ offline and store them for later use even if this storage is less protected than the more sensitive long-term private key.

Lemma 5 *Protocol HOMQV remains secure even when making the ephemeral DH exponents y available to the attacker via state reveal queries. Moreover, even sessions for which y is known remain secure (formally, the attacker is allowed to choose a test session for which it knows y).*

Proof: We show an even stronger property: Let $\hat{B}$ be an honest party and assume that $\mathcal{M}$ gets to see all pairs $\{(y_i, Y_i = g^{y_i})\}$ to be used by $\hat{B}$ for all its sessions at the onset of the protocol run. We claim that all of $\hat{B}$'s unexposed sessions (where learning the y_i exponent is *not* considered exposure) are still secure. In other words, even if the test session chosen by $\mathcal{M}$ is one of these outgoing sessions of $\hat{B}$ (or one of the matching sessions at the peer) the attacker cannot win the distinguishing game with non-negligible advantage. The proof is essentially the same as the one for Theorem 4; all we need to observe is that in the simulation in that proof the forger $\mathcal{F}$ chooses (and hence knows) all exponents y corresponding to outgoing DH values Y at $\hat{B}$. Hence, $\mathcal{F}$ can provide these values to $\mathcal{M}$, even before they are used.

For this, the description of $\mathcal{F}$ when simulating a Send query to $\hat{A}$ or $\hat{B}$ needs to be slightly changed. Specifically, it needs to use its DDH oracle to test all previous query to H, just as it is done for a Receive query. This is needed here since $\mathcal{M}$, knowing y from the start, could have made a query that depends on this value even before it was used by $\hat{B}$. □

5.4 Sender's Forward Secrecy

We show that HOMQV ensures *sender's forward secrecy* against passive attackers. Specifically, we show that the exposure of the private key of a party does not compromise any of the past sessions where that party acted as sender. Moreover, this is the case even for sessions established after the disclosure of the private key, provided that the attacker does not control the session's Y or knows its exponent y. This property, however, does not guarantee forward secrecy for sessions where the attacker actively chose the value Y prior to the private key exposure. For example, an attacker can choose y, set $Y = g^y$, and activate party $\hat{A}$ as receiver with incoming Y and sender's identity $\hat{B}$. Later, if the attacker finds $\hat{B}$'s private key, it can compute the value of the session key generated by $\hat{A}$ using Y. In other words, the protocol provides *weak forward secrecy* against passive attackers, but not full PFS against active attacks. Fortunately, we will see below that one can slightly change HOMQV by adding a (key confirmation) field to the message from $\hat{B}$ to $\hat{A}$ and then achieve full sender's forward secrecy.

Lemma 6 *HOMQV enjoys sender's forward secrecy against passive attackers.*

Proof: (sketch) Let $\hat{B}$ be an honest party and assume that at some point $\mathcal{M}$ learns $\hat{B}$'s private key b. We claim that even in this case $\mathcal{M}$ cannot win the test session experiment for any session in which $\hat{B}$ acted as sender provided that the outgoing value Y for the session was indeed generated by $\hat{B}$ and that the session was not exposed via a session-key query or via a state-reveal query. We informally outline the proof of this property.

For contradiction we assume an attacker $\mathcal{M}$ that wins the test-session experiment with non-negligible advantage in a session as above after learning the sender's private key. We use $\mathcal{M}$ to build an algorithm $\mathcal{S}$ that solves CDH given a decisional oracle as in the Gap-DH setting. Let $X, Y \in_{\mathrm{R}} G$ be an instance of the CDH problem. $\mathcal{S}$ will run a simulation similar to the one $\mathcal{F}$ runs in the proof of Theorem 4; in particular, it will try to guess the peers to the test session (call $\hat{B}$ the guess for sender and $\hat{A}$ the guess for receiver) as well as which of the sessions between $\hat{B}$ and

$\hat{A}$ will be chosen as the test. $\mathcal{S}$ chooses public keys for all honest parties *including* $\hat{B}$ *but excluding* $\hat{A}$. For $\hat{A}$ it sets the public key to X. The rest of the simulation is similar except that $\hat{B}$'s private key is known (chosen by $\mathcal{S}$) so $\hat{B}$ is treated as any other uncorrupted party. The simulation of $\hat{A}$ for whom the private key is unknown is same as in the proof of Theorem 4 (in particular, it uses the decisional oracles). When the guessed test session is activated at $\hat{B}$, $\mathcal{S}$ will use the value Y from the CDH input as the outgoing value from $\hat{B}$. When $\mathcal{M}$ corrupts $\hat{B}$, $\mathcal{S}$ provides it with $\hat{B}$'s private key b which $\mathcal{S}$ knows. As before, to win the test experiment $\mathcal{M}$ needs to be able to compute with non-negligible probability the value $\sigma = (YB^e)^a$ which $\mathcal{S}$ learns from the H-queries. From σ, $\mathcal{S}$ computes $Z = Y^a$ (where $A = g^a$) by setting $Z = \sigma/A^{eb}$. Since $Z = DH_g(X, Y)$, $\mathcal{S}$ solved the CDH challenge contradicting the Gap-DH assumption. □

5.4.1 Full Sender's Forward Secrecy

As said, HOMQV does not achieve forward secrecy for sessions activated at a receiver $\hat{A}$ where the incoming value Y was chosen by the attacker $\mathcal{M}$. To fix this we need to assure that $\hat{A}$ will not accept an incoming Y that has not been generated by the purported sender. For this, add to the protocol's message *key confirmation* field. First, we define a key derivation step that from the key K, as defined in HOMQV, derives two keys K^*, K_a. The former is output as the session key while K_a is used as a key to a MAC function. The key confirmation field (included in the message sent from $\hat{B}$ to $\hat{A}$) is a tag $\text{MAC}_{K_a}(1)$ (where 1 can be replaced by any fixed message that is publicly determined by the protocol flows). It is not hard to see that in this way a session at a honest receiver will only be established if the MAC was successful hence proving that Y was generated by the claimed sender. (Formally, one shows that if a MAC verification succeeds for a value not generated by the claimed sender then the security of the original HOMQV protocol is broken, or more specifically, that a successful XCR forgery occurs.) We have:

Lemma 7 *The modified HOMQV protocol with $\hat{B}$'s key confirmation provides full sender's forward secrecy (against active attacks).*

6 Extensions in the Interactive Setting

In settings where interaction between server and client is possible, the HOMQV protocol smoothly "extends up" to provide better security at a very low cost. Replay attacks can be prevented simply by having $\hat{A}$ send a nonce to $\hat{B}$ in the first flow, and then incorporate that nonce in the final hash calculation, setting $K = H(\sigma, \hat{B}, \hat{A}, Y, n)$ (with n the nonce sent by $\hat{A}$). This simple variant does not offer receiver forward secrecy or protection against KCI attacks, but it does prevent replay attacks without incurring any additional computational cost. Note also that by making the nonce n default to null in an implementation of this variant, we get "transparent" support also for the non-interactive HOMQV.

To get also receiver forward secrecy and protection against KCI attacks, we can move up to two- or three-pass HMQV, where $\hat{A}$ sends $X = g^x$ to $\hat{B}$ and σ is computed as $\sigma = (XA^d)^{f(y+eb)} = (YB^e)^{f(x+da)} = g^{f(x+da)(y+eb)}$, with $d = \bar{H}(X, \hat{B})$ and $e = \bar{H}(Y, \hat{A})$. The price we pay for this added security over the previous variant is another 1/2 exponentiation for the sender (to compute (XA^d)) and another full exponentiation for the receiver (to compute $X = g^x$).

References

[1] Abdalla, M., Bellare, M., Rogaway, P.: The Oracle Diffie-Hellman Assumptions and an Analysis of DHIES. In: Topics in Cryptology - CT-RSA '01. Lecture Notes in Computer Science, vol. 2020, pp. 143–158. Springer (2001)

[2] Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: 42nd Annual Symposium on Foundations of Computer Science - FOCS'01. pp. 136–145. IEEE (2001)

[3] Canetti, R., Krawczyk, H.: Analysis of key-exchange protocols and their use for building secure channels. In: Advances in Cryptology - EUROCRYPT'01. Lecture Notes in Computer Science, vol. 2045, pp. 453–474. Springer (2001)

[4] Canetti, R., Krawczyk, H.: Security analysis of ike's signature-based key-exchange protocol. In: Advances in Cryptology - CRYPTO'02. Lecture Notes in Computer Science, vol. 2442, pp. 143–161 (2002)

[5] Dent, A.W.: Hybrid cryptography. Cryptology ePrint Archive, Report 2004/210 (2004), `http://eprint.iacr.org/`

[6] Dent, A.W.: Hybrid signcryption schemes with insider security. In: Information Security and Privacy - ACISP'05. Lecture Notes in Computer Science, vol. 3574, pp. 253–266. Springer (2005)

[7] Dent, A.W.: Hybrid signcryption schemes with outsider security. In: Information Security - ISC'05. Lecture Notes in Computer Science, vol. 3650, pp. 203–217. Springer (2005)

[8] Gennaro, R., Halevi, S.: More on key wrapping. In: Selected Areas in Cryptography - SAC'09. Lecture Notes in Computer Science, vol. 5867, pp. 53–70. Springer (2009)

[9] Gjøsteen, K., Kråkmo, L.: Universally composable signcryption. In: Lopez, J., Samarati, P., Ferrer, J. (eds.) Public Key Infrastructure. Lecture Notes in Computer Science, vol. 4582, pp. 346–353. Springer (2007)

[10] Gorantla, M., Boyd, C., Gonzlez Nieto, J.: On the connection between signcryption and one-pass key establishment. In: Galbraith, S. (ed.) Cryptography and Coding, 11th IMA International Conference. Lecture Notes in Computer Science, vol. 4887, pp. 277–301. Springer (2007)

[11] IEEE 1363a-2004: IEEE standard specifications for public-key cryptography - amendment 1: Additional techniques

[12] Krawczyk, H.: HMQV: A high-performance secure diffie-hellman protocol. In: Advances in Cryptology - CRYPTO'05. Lecture Notes in Computer Science, vol. 3621, pp. 546–566. Springer (2005)

[13] Menezes, A.: Another look at HMQV. `http://eprint.iacr.org/2005/205` (2005)

[14] Menezes, A., van Oorschot, P., Vanstone, S.: Handbook of Applied Cryptography. CRC Press (1996)

[15] Rogaway, P., Shrimpton, T.: A provable-security treatment of the key-wrap problem. In: Advances in Cryptology - EUROCRYPT'06. Lecture Notes in Computer Science, vol. 4004, pp. 373–390. Springer (2006)

[16] Shoup, V.: ISO 18033-2: An emerging standard for public-key encryption. Available at `http://shoup.net/iso/`

[17] Zheng, Y.: Digital signcryption or how to achieve cost(signature & encryption) $\ll$ cost(signature) + cost(encryption). In: Advances in Cryptology - CRYPTO'97. Lecture Notes in Computer Science, vol. 1294, pp. 165–179. Springer (1997)

A A Cofactor-Free Variant of HOMQV

As we said in the remark before Section 3.1, if we remove the cofactor f from HOMQV then the sender needs to check that the receiver's public key is in the subgroup G, and the receiver needs to check that the value YB^e is in the subgroup G. This is apparent in the proof of Theorem 4, in that we must ensure that the DDH oracle is only called on inputs that are indeed elements in the subgroup G. The modified protocol then becomes as follows:

Sender $\hat{B}$, sending to receiver $\hat{A}$. *$\hat{B}$ verifies that $\hat{A}$'s public key A is in the subgroup G.* If so, then $\hat{B}$ chooses a fresh random y and sends $Y = g^y$ to $\hat{A}$. $\hat{B}$ also computes $e = \bar{H}(Y, \hat{A})$ and $\sigma = A^{y+eb}$, and sets the session key to $K = H(\sigma, \hat{B}, \hat{A}, Y)$.

Receiver $\hat{A}$, receiving value Y from sender $\hat{B}$. $\hat{A}$ computes $e = \bar{H}(Y, \hat{A})$ and $X = YB^e$, and *verifies that $X \in G$.* If so then $\hat{A}$ computes $\sigma = X^a$ and sets the session key to $K = H(\sigma, \hat{B}, \hat{A}, Y)$.

On the sender's test $A \in G$.

It is not hard to see that the test $X \in G$ at the receiver's is needed to prevent a small subgroup attack against the receiver's private key. However, the test $A \in G$ at the sender's is *only needed if we require "y-security"*; indeed if y is known to the attacker a small group attack against $\hat{B}$'s private key b is possible. If we assume y to remain secret then the test is not needed as proven next.

Lemma 8 *In an attack scenario where y is not available to the attacker, HOMQV without cofactor exponentiation is secure (with sender's forward secrecy) even if the sender $\hat{B}$ does not check the receiver's public key A to be in G.*

Proof: The argument follows, with some modifications, the proof of Theorem 4 which uses an HOMQV-attacker $\mathcal{M}$ to build an XCR-forger $\mathcal{F}$ where the inputs to $\mathcal{F}$ are a signer's public key B and challenge X. The guessed peers to the test session are denoted $\hat{B}$ (the sender) and $\hat{A}$ (receiver) and their public keys are set to B and $A = X$, respectively.

The simulation proceeds exactly as in the previous proof except for the case where $\hat{B}$ acts as sender with a corrupted peer $\hat{C}$. In this case, instead of choosing a random Y as the outgoing value from $\hat{B}$, $\mathcal{F}$ chooses random $s \in Z_q$ and $e \in \{0,1\}^{|q|/2}$ and sets $Y = g^s/B^e$ and $\bar{H}(Y, \hat{C}) = e$. The session key $H(\sigma, \hat{B}, \hat{C}, Y)$ is set to a random $K \in \{0,1\}^k$ (if Y equals a previous outgoing value from $\hat{B}$, $\mathcal{F}$ aborts). Note that the σ value for this session equals C^s which $\mathcal{F}$ knows and hence it can answer K if at any point $\mathcal{M}$ queries H on input $(\sigma, \hat{B}, \hat{C}, Y)$.

Consider now the test session chosen by $\mathcal{M}$ that we denote $(\hat{B}, \hat{A}, Y)$ (if the test session has peers other than $\hat{B}$ as sender and $\hat{A}$ as receiver, $\mathcal{F}$ aborts). Let K denote the value for the session key set by $\mathcal{F}$ for this session. When $\mathcal{M}$ performs the test session query, $\mathcal{F}$ flips a coin and returns K with probability 1/2 and a random independent K' with probability 1/2. Since $\mathcal{M}$ is assumed to win the test session with non-negligible advantage then with non-negligible probability $\mathcal{M}$ queries $H(\sigma, \hat{B}, \hat{A}, Y)$ for $\sigma = (YB^e)^a$. In other words, $\mathcal{M}$ generates a valid signature of $\hat{B}$ on message $\hat{A}$ using Y and challenge $A = X$. To show that this signature qualifies as a successful forgery we need to show that no other signature generated by $\hat{B}$ (i.e., generated by $\mathcal{F}$ on behalf of $\hat{B}$) in the above run had the pair $(Y, \hat{A})$. We consider three cases:

1. The value Y was output by $\hat{B}$ in an activation with peer $\hat{A}$ (i.e., the session $(\hat{B}, \hat{A}, Y)$ exists at $\hat{B}$). In this case, $\hat{B}$ never computed a signature with values $(Y, \hat{A})$ since the only signatures generated by $\hat{B}$ are those for sessions with a corrupted peer and $\hat{A}$ is uncorrupted (or else $\mathcal{F}$ would have aborted).

2. The value Y was output by $\hat{B}$ with a peer $\hat{C} \neq \hat{A}$. In this case $\hat{B}$ might have generated a signature on $\hat{C}$ with value Y (this is indeed the case if $\hat{C}$ is corrupted) but never a signature on $\hat{A}$ with value Y.

3. The value Y was never output by $\hat{B}$. Therefore, $\hat{B}$ could have not generated any signature with value Y.

□

Note. The above proof does not require the use of a DDH oracle with respect to the sender's public key B, showing that when y is assumed to be secret such oracle (used in the proof of Theorem 4) is unnecessary.