HMQV: A High-Performance Secure Diffie-Hellman Protocol *

Hugo Krawczyk[†]

July 5, 2005

Abstract

The MQV protocol of Law, Menezes, Qu, Solinas and Vanstone is possibly the most efficient of all known authenticated Diffie-Hellman protocols that use public-key authentication. In addition to great performance, the protocol has been designed to achieve a remarkable list of security properties. As a result MQV has been widely standardized, and has recently been chosen by the NSA as the key exchange mechanism underlying *"the next generation cryptography to protect US government information"*.

One question that has not been settled so far is whether the protocol can be proven secure in a rigorous model of key-exchange security. In order to provide an answer to this question we analyze the MQV protocol in the Canetti-Krawczyk model of key exchange. Unfortunately, we show that MQV fails to a variety of attacks in this model that invalidate its basic security as well as many of its stated security goals. On the basis of these findings, we present HMQV, a carefully designed variant of MQV, that provides the same superb performance and functionality of the original protocol but for which all the MQV's security goals can be formally proved to hold in the random oracle model under the computational Diffie-Hellman assumption.

We base the design and proof of HMQV on a new form of "challenge-response signatures", derived from the Schnorr identification scheme, that have the property that both the challenger and signer can compute the *same* signature; the former by having chosen the challenge and the latter by knowing the private signature key.

Revision. In `http://eprint.iacr.org/2005/205`, Menezes [32] describes some shortcomings in our analysis that lead to the need for a prime-order verification of public DH values in the protocol. Some of Menezes's claims are correct and some other are not. We keep the originally posted paper here but add a *Preface* section (preceding the introduction) that briefly explains these findings and their implications to our results. In essence, the provability of HMQV and its security superiority relative to MQV remain valid; even computation-wise, after adding the verification steps where needed, HMQV is as efficient as (and in some cases even more efficient than) MQV.

[*] An extended abstract of this paper appears in Crypto'05.

[†] IBM T.J.Watson Research Center, PO Box 704, Yorktown Heights, NY 10598, USA. Email: hugo@ee.technion.ac.il

Contents

Preface

Recently, Menezes [32] has claimed that contrary to what is stated in this paper the proven security of HMQV depends on the execution of an explicit verification that DH public values belong to the prime-order group G used by the protocol (below we refer to these tests as G-*tests*). Menezes's claim is not true for the core HMQV protocol as specified in Section 5.1 and proven in Sections 5 and 6. However, G-tests are needed when requiring that the protocol remain secure even when ephemeral exponents are revealed (Section 7.1) and for the one-pass protocol of Section 9.

The reason that G-tests are *not* required for the basic security of HMQV is that, as stated in this paper, the security of XCR signatures (Section 4), on which HMQV is based, holds even if DH values are chosen by the attacker as arbitrary elements of a supergroup G' that contains G. In particular, Menezes's claim that the proof of XCR signatures is flawed is incorrect. On the other hand, he is correct in that the proof of HCR signatures (Lemma 27) implicitly assumes that the challenge X lies in the subgroup G. As a consequence, when applying HCR signatures in the context of HMQV one must require that public DH values (static and ephemeral) be tested for membership in G. Specifically, this is needed for the case in which ephemeral DH exponents are kept in the session state (and then vulnerable to state-reveal queries) as in Section 7 and for the one-pass protocol of Section 9.

This all boils down to the following conclusions. When using the one-pass protocol one should always perform G-tests. When running the 2-message or 3-message HMQV protocol one can dispense completely with these tests but then the protocol may become insecure (this depends on the order of the supergroup G') if secret ephemeral exponents are revealed. Thus, if one wants to ensure security in the presence of the disclosure of ephemeral exponents then G-tests need to be required. This provides a clear performance-security trade-off: Perform G-tests and get a stronger security assurance, or do not perform these tests but make sure to protect your ephemeral exponents as well as your private key (such a protection is assumed, for example, for DSA-like signatures).

Finally, when compared to MQV, HMQV retains all the security advantages stated and proven in this paper. HMQV is also superior to MQV in performance in cases that group membership tests are not required and is otherwise of the same efficiency as MQV (in MQV an exponentiation whose cost is equivalent to a G-test is always performed).

A personal perspective. I would like to thank Alfred Menezes for identifying the oversight in the HCR proof and the need for group membership verification in the one-pass protocol. At the same time, I must strongly disagree with the attempt in [32] to discredit the effort of the cryptographic community dedicated to improving our understanding and design of protocols. True, we make mistakes (and I do not justify my own); and proofs (even if correct) are never stronger than the model and assumptions they are based on. But with all its imperfection, this form of analysis is an essential tool for gaining confidence in the soundness of a cryptographic design. Moreover, as clearly shown here, the proof process itself serves as a guide in choosing the right design elements.

At a time when we demand the best (almost perfect) security from basic encryption and hash functions, and having witnessed the effects of initially-mild attacks, we can only hope that the applied-cryptography community and its representing standard bodies will see formal analysis as a requirement, and main source of confidence, when adopting protocols for wide use. These analyses can (and must) be verified by the community at large (in contrast, ad-hoc designs do not even provide the "luxury" of judging well-defined security properties). This is all the more significant in the case of a protocol such as MQV which is not only intended for wide commercial use but also to protect "classified or mission critical national security information" [36].

1 Introduction

The classic Diffie-Hellman (DH) key-exchange protocol (see Figure 1) that marked the birth of modern cryptography has since been one of the main pillars of both theory and practice of cryptography. While the basic protocol as originally proposed is believed to be secure against an eavesdropping-only attacker, the quest for an "authenticated Diffie-Hellman" protocol that resists active, man-in-the-middle, attacks has resulted in innumerable ad-hoc proposals, many of which have been broken or shown to suffer from serious weaknesses. Fortunately, with the development of rigorous security models for key exchange in the last years, we are now in a much better position to judge the security of these protocols as well as to develop designs that provably withstand realistic active attacks.

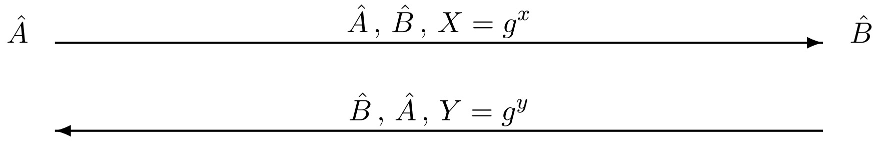


Figure 1: The basic (unauthenticated) Diffie-Hellman protocol

In addition to the need for sound security, the many practical applications of key exchange have driven designers to improve on the performance cost associated with authentication mechanisms, especially those based on public key. One ambitious line of investigation, initiated by Matsumoto, Takashima and Imai in 1986 [30], is to design DH protocols whose communication is identical to the basic DH protocol (i.e., no explicit authentication added except for the possible transmission of PK certificates), yet they are implicitly authenticated by the sole ability of the parties to compute the resultant session key (i.e., rather than agreeing on the key g^{xy}, the parties would agree on a key that combines g^x, g^y with their public/private keys). Not only can this approach generate protocols that are very efficient communication-wise, but the combination of authentication with the key derivation procedure can potentially result in significant computational savings. For these reasons, several of these "implicitly authenticated" protocols have been standardized by major national and international security standards.

Of these protocols, the most famous, most efficient and most standardized is the MQV protocol of Law, Menezes, Qu, Solinas and Vanstone [33, 29]. This protocol has been standardized by many organizations, e.g. [2, 3, 20, 21, 35], and has recently been announced by the US National Security Agency (NSA) as the key exchange mechanism underlying "the next generation cryptography to protect US government information" (which includes the protection of "classified or mission critical national security information") [36]. Indeed, MQV appears to be a remarkable protocol that not only is the most efficient and versatile authenticated DH protocol in existence, but it has also been designed to satisfy an impressive array of security goals.

Yet, in spite of its attractiveness and success, MQV has so far eluded any formal analysis in a well-defined model of key exchange. The present work was initially motivated by the desire to provide such an analysis. Our findings, however, have been disappointing: we found that when formally studied virtually none of the stated MQV goals can be shown to hold (specifically, we carried this study in the computational key exchange model of Canetti and Krawczyk [11]). This raises clear concerns about the security of the protocol and triggers a natural question: Do we have a replacement for MQV with the same superb performance and versatility but for which the MQV security goals can be guaranteed in a well analyzed, provable way?

The main contribution of this paper is in identifying the various analytical shortcomings of the MQV design and proposing a "hashed variant" of the protocol, which we call HMQV, that provides the same (almost optimal) performance of MQV but also delivers, in a provable way, the original security goals of MQV (and even more).

Note on groups and notation. All the protocols and operations discussed in this paper assume a cyclic group G of prime order q generated by a generator g. We denote by $|q|$ the bit length of q (i.e., $|q| = \lceil \log_2 q \rceil$), and use this quantity as an implicit security parameter. The parameters G, g and q are assumed to be fixed and known in advance to the parties (this is usually the case in practice, e.g., [19]; alternatively, one could include these values in certificates, etc.). We use the multiplicative representation of group operations but our treatment is equally applicable to additive (prime order) groups such as elliptic curves. In our protocols, public keys (denoted by upper case letters) are elements in the group G, and the private keys (denoted by the corresponding lower case letters) are elements in Z_q. For example, to a public key $A = g^a$ corresponds a private key a. The party having A as its public key will be denoted by $\hat{A}$ (in general, the "hat notation" is used to denote the identities of parties in the protocol, possibly including the party's PK certificate). All parties in the protocol are probabilistic polynomial-time machines, including the attacker. The latter is denoted by $\mathcal{M}$ (where $\mathcal{M}$ stands for "malicious" or, paraphrasing the Alice and Bob tradition, $\mathcal{M}$ may stand for "Malice" or "Mob").

1.1 The MQV Protocol

Key Computation in the MQV and HMQV Protocols

Both protocols: $\hat{A}$ and $\hat{B}$ run a basic Diffie-Hellman exchange (Figure 1)

$\hat{A}$ computes $\sigma_{\hat{A}} = (YB^e)^{x+da}$, $\hat{B}$ computes $\sigma_{\hat{B}} = (XA^d)^{y+eb}$

Both parties set $K = H(\sigma_{\hat{A}}) = H(\sigma_{\hat{B}})$

MQV: $d = \bar{X} \overset{\text{def}}{=} 2^\ell + (X \bmod 2^\ell)$ $e = \bar{Y} \overset{\text{def}}{=} 2^\ell + (Y \bmod 2^\ell)$ $\ell = |q|/2$.

HMQV: $d = \bar{H}(X, \hat{B})$, $e = \bar{H}(Y, \hat{A})$,

Figure 2: Computation of the session key K in each of the two protocols ($A = g^a$ and $B = g^b$ are $\hat{A}$'s and $\hat{B}$'s public keys, respectively.)

The communication in the MQV protocol is identical to the basic unauthenticated DH protocol depicted in Figure 1 except that the identities $\hat{A}, \hat{B}$ may include a public-key certificate. The computation of the session key is shown in Figure 2 where: party $\hat{A}$ possesses a long-term private key $a \in Z_q$ and corresponding public key $A = g^a$, $\hat{B}$'s private/public key pair is $(b, B = g^b)$, and the ephemeral DH values exchanged in the protocol are $X = g^x, Y = g^y$ (x, y chosen by $\hat{A}, \hat{B}$, respectively). The computation of the session key also uses the values $d = \bar{X}$ and $e = \bar{Y}$, where $\bar{X} = 2^\ell + (X \bmod 2^\ell)$ and $\bar{Y} = 2^\ell + (Y \bmod 2^\ell)$ for $\ell = |q|/2$. The computation of the session key by $\hat{A}$ (and similarly by $\hat{B}$) involves the exponentiations $X = g^x$, B^e, and $(YB^e)^{x+da}$. Note, however, that e is of length $|q|/2$ and hence B^e counts as "half exponentiation" (i.e. half the number of modular multiplication relative to a regular exponentiation of g). Also, note that $X = g^x$ can be pre-computed. This sums up to an impressive performance: same communication as the basic DH protocol and just half exponentiation more than the basic protocol, i.e. *a mere 25% increase in computation to achieve an authenticated exchange!* This is significantly better than any of the proven DH protocols that rely on digital signatures or public key encryption for authentication (which involve more expensive operations and increased bandwidth), and is also the most efficient of the implicitly-authenticated DH protocols (the closest are the "Unified Model" protocols [8, 23] that require three full exponentiations and offer substantially less security features – see Section 1.6).

1.2 Stated Security Goals of the MQV Protocol

The designers of MQV clearly, albeit informally, stated the security goals behind the MQV design (see [33, 29] and related publications). This includes the resistance to a variety of explicit attack strategies such as guessing attacks, impersonation attacks, known-key attacks, key-compromise impersonation (KCI) attacks, and the provision of perfect forward secrecy (PFS).

While resistance to guessing attacks and impersonation attacks are basic and obvious security requirements, it is worth expanding on the meaning of the other attacks. They all represent realizations of the same fundamental security principle: a good security system is not one that denies the possibility of failures but rather one designed to confine the adverse effects of such failures to the possible minimum.

In the case of known key attacks, one is concerned with the realistic possibility that some session-specific information, such as a session key (or the ephemeral secrets that led to the computation of that key), will leak to an attacker. This can happen in a variety of ways ranging from the simple mishandling of information to a temporary break-in into a computer system or the malicious action of an insider. In this case, one does not expect the exposed session to remain secure, but a well-designed key-exchange protocol needs to guarantee that such a failure will *only* affect the specific compromised session. Other sessions, by the same or other parties, should not be endangered by this leakage. The resistance to known-key attacks enforces other basic security principles as well; most importantly, that keys from different sessions should be fully "computationally independent" (i.e., from learning one session key nothing can be implied about the value of other session keys).

The properties of PFS and KCI resistance are also concerned with limiting the effects of eventual failures, in this case the disclosure of long-term keys. Clearly, the discovery by an attacker $\mathcal{M}$ of the long-term authentication key of party $\hat{A}$ allows $\mathcal{M}$ to impersonate $\hat{A}$ and establish its own sessions in the name of $\hat{A}$. A protocol is said to have PFS if session keys established (and deleted from memory) before the compromise of the long-term key cannot be recovered (even with the use of this key). In the case of KCI, the question is whether the knowledge of $\hat{A}$'s private key allows $\mathcal{M}$ not only to impersonate $\hat{A}$ to others but also *to impersonate other, uncorrupted, parties to* $\hat{A}$. A protocol that prevents this form of "reverse impersonation" is said to resist KCI attacks. In other words, in such a protocol the only way $\mathcal{M}$ can take advantage of the knowledge of $\hat{A}$'s private key is by active impersonation of $\hat{A}$. Any session established by $\hat{A}$, without being actively controlled by $\mathcal{M}$, remains secure. Resistance to KCI attacks is a very significant security property that has added much to the attractiveness of MQV as it is not offered by other implicitly-authenticated protocols, such as the unified-model protocols of [8, 23] (see Section 1.6), that use the static DH key g^{ab} for authentication (this key functions as a long-term shared key and hence cannot resist a KCI attack).

Another important robustness consideration for any Diffie-Hellman protocol is its resistance to the leakage of ephemeral secret DH exponents. While this property has not been addressed directly by the designers of MQV, we consider it a prime security concern and therefore carefully study it in the context of HMQV (see Section 1.4).

Note: One assurance that MQV is *not* designed to provide is that a party completing a session with (assumed) peer $\hat{B}$ knows for certain that this peer computed the session key or even that the peer was alive during this exchange. Such an assurance cannot be provided (at least not for the responder) by any 2-message protocol. More importantly, while this may be a desirable operational property in some applications (in which case the protocol needs to be augmented to three rounds as we discuss later), it is not essential for securing communications as demonstrated in [11]. The essential security property is that $\hat{A}$ is guaranteed that if the assumed peer $\hat{B}$ is uncorrupted then $\hat{B}$ is the only party that may possibly know K.

1.3 Weaknesses of the MQV Protocol

In spite of the ambitious security goals described above, it turns out that when casting these goals in a well-defined formal setting as the one in [11], the MQV protocol falls short of delivering most of its intended security. Here we summarize some of these findings. (See Section 3.2 for a detailed presentation of MQV's shortcomings.)

Group representation attacks. We first observe that MQV's security is strongly susceptible to the specific way the group elements are represented in the protocol. We show how some representations render the protocol totally insecure. While ordinary group representations may not have such an extreme effect on the security of the protocol, this result shows that any attempt at proving MQV would need to involve restricted group representations. Moreover, the inherent weaknesses of the protocol discussed below show that the protocol cannot be proven secure even for specific groups.

UKS attacks. We study the vulnerability of the protocol to "unknown key share" (UKS) attacks which were also listed as a security consideration in MQV. A successful UKS attack [16] is one in which two parties compute the same session key but have different views of who the peer to the exchange was (this attack represents both an authentication failure as well as a vulnerability to known-key attacks). Originally, it was thought that MQV (at least when the registrants of public keys are required to prove "possession" of the corresponding private keys) was immune to these attacks; later it was shown by Kaliski [24] that even with such proofs of possession MQV fails to a UKS attack. Since then it has been believed that augmenting the protocol with a "key confirmation" step (which adds a third message to the protocol) would solve the problem. Here we show that this is *not* the case. Indeed, the 3-message variant of MQV is still vulnerable to this form of attack if the attacker can learn ephemeral session-state information for sessions other than the session being attacked.

Lack of PFS. MQV does not provide Perfect Forward Secrecy (PFS). This, however, is not just a failure of MQV but it's an inherent limitation of implicitly-authenticated 2-message protocols based on public-key authentication (and which do not rely on a previously established shared state between the parties). Indeed no such protocol can provide PFS. We present a generic attack against any such protocol where an active attacker $\mathcal{M}$ causes the establishment of a session key K at party $\hat{A}$ with peer $\hat{B}$ such that a later corruption of $\hat{B}$ (even after K was erased) allows $\mathcal{M}$ to find K.

KCI attacks. Since MQV is susceptible to basic authentication attacks even when the private key of the victim is not known to the attacker, then KCI resistance cannot be satisfied. Yet, it is interesting to see explicit KCI attacks that take advantage of the knowledge of such private key. We show such an attack against MQV in the case that the attacker has access to the ephemeral values σ from which the session key is computed. This serves to motivate two design principles in HMQV: (i) the essential role of the hashing of σ for session key derivation (in MQV this hashing is recommended but separated from the core specification of the protocol [29])[1]; and (ii) the care required in handling ephemeral information (especially when an attacker may access such information).

Prime-order checks. Our description in Figure 2 omitted an element from the session-key computation in MQV: In case where the group G generated by g is a subgroup of a larger group G' (which is the case in typical mod p and elliptic curve groups), MQV specifies that the key be computed as $K = H(\sigma)$ for $\sigma = (\sigma_{\hat{A}})^h = (\sigma_{\hat{B}})^h$ ($\sigma_{\hat{A}}$ and $\sigma_{\hat{B}}$ are defined in Figure 2) where h denotes the co-factor $|G'|/|G|$. This measure is used to ensure that the value σ belong to the group G, and has been added to MQV as a safeguard against potential attacks resulting from the lack of explicit authentication of the DH values, e.g., small-group attacks. We note, however, that this addition is of no help against the vulnerabilities mentioned above (and as we will see is not needed to provide security in HMQV). Moreover, lacking a proof that the above counter-measure really works, several standards defining MQV, as well as various descriptions of the protocol in the literature, often specify (or at least recommend) that the parties to MQV explicitly check that the DH value presented by the peer is of prime order q. This adds a costly extra exponentiation to each peer and takes significantly from the almost-optimal performance of MQV. As we will see, HMQV "provably dispenses" of the need for this costly check.

1.4 The HMQV Protocol and its Proven Security

Description of the HMQV Protocol. The HMQV protocol ("H" is for Hash) is a simple but powerful variant of MQV. As in MQV, its communication is identical to the basic DH exchange (Figure 1) with the possible addition of certificates. The computation of the session key, shown in Figure 2, differs from MQV's in the computation of the values d and e which involves the hashing of the party's own DH value and the peer's identity. The output of this hash is $\ell = |q|/2$ bits. In addition, HMQV *mandates* the hashing of the values $\sigma_{\hat{A}} = \sigma_{\hat{B}}$ into k-bit keys where k is the length of the desired session key. We denote the hash function with ℓ bits of output by $\bar{H}$ and the one with k bits by H. In practice the same hash function can be used with different output lengths. (As a mnemonic, the bar in $\bar{H}$ indicates that the output of the function is used as an exponent).

From this description one can see that HMQV preserves the outstanding performance of MQV (both in terms of communication and computation). At the same time, *HMQV overcomes all the mentioned security shortcomings of MQV to the largest possible extent in a 2-message protocol.* We prove that in the random oracle model [5], and under the Computational Diffie-Hellman (CDH) assumption [15], the protocol is secure in the Canetti-Krawczyk security model [11]. In particular, this establishes the security of the protocol against impersonation attacks, known-key attacks, and UKS attacks. We also prove the resistance of HMQV to KCI attacks (which we formally define) under the same assumptions.

Furthermore, HMQV enjoys an additional performance advantage in that it provably dispenses of the need for costly prime-order tests on the DH values transmitted in the protocol. Indeed, our proof shows that the only way an attacker can benefit from the choice of rogue DH values is by choosing these to be zero, and thus a simple non-zero check is all is required (hence, there is no need for prime-order tests or for the co-factor h used in MQV).

[1] The MQV paper is somewhat ambiguous about the need to hash σ (see the end of Sections 1 and 5 in [29]). In particular, this hashing is not viewed as essential to the security of the protocol but as a possible safeguard against potential weak bits in σ (which is not the source of weakness in this example, but rather the malleability of σ is.)

Regarding forward secrecy, we said earlier that PFS cannot be achieved by any implicitly authenticated 2-message protocol, including HMQV. Yet, the following limited forward secrecy property holds for HMQV: any session key established without the active intervention of the attacker (except for eavesdropping the communication) is guaranteed to be irrecoverable by the attacker once the session key is erased from memory. This is the case even if the attacker knew the private keys of both peers when the session was established.

For applications that require full PFS we present a 3-message variant of HMQV which adds a third message and a MAC computation by each party (see Figure 7 in page 55) and guarantees full PFS. This 3-message protocol, called HMQV-C, also provides "key confirmation" to both parties, i.e., the assurance that the assumed peer indeed participated in the protocol and that it computed the same session key. Another advantage of HMQV-C is that it can be proven secure in the stronger *universally composable* (UC) KE security model of [13] which ensures the security of key-exchange protocols when run concurrently with other applications. (HMQV satisfies the weaker "relaxed UC" definition from [13].) We note that while HMQV-C requires an extra message, its computational cost is essentially the same as HMQV as the MAC computation is negligible relative to the exponentiation cost. On the other hand, [23] note that 2-message symmetric protocols such as the basic HMQV allow for simultaneous initiation of a session by both $\hat{A}$ and $\hat{B}$, a desirable property in some network settings.

Another variant of HMQV is a one-pass authenticated key-exchange protocol in which $\hat{A}$ sends a single message to $\hat{B}$ after which both parties share a secret key. We show also this protocol to be secure (under CDH and in the random oracle model) in a security model adapted from [11] to one-pass protocols (the only difference is that we cannot prevent the adversarial replay of a message from $\hat{A}$ to $\hat{B}$ and, of course, cannot provide PFS). In particular, this one-pass protocol provides the functionality of public-key based deniable authentication as well as an authenticated CCA encryption scheme (in the random oracle model) in a more efficient way than existing alternatives.

An important security consideration not discussed by the authors of MQV is the resilience of the protocol to the disclosure of the secret exponent x corresponding to an ephemeral (session-specific) DH value $X = g^x$. This is a prime concern for any Diffie-Hellman protocol since many applications will boost protocol performance by pre-computing ephemeral pairs $(x, X = g^x)$ for later use in the protocol (this may apply to low-power devices as well as to high-volume servers). In this case, however, these stored pairs are more vulnerable to leakage than long-term static secrets (the latter may be stored in a hardware-protected area while the ephemeral pairs will be typically stored on disk and hence more available to a temporary break or to a malicious user of the system). We prove that HMQV's security is preserved even in the presence of the leakage of ephemeral secret DH exponents (see Section 1.6 for comparison to other work). For this property (and only for it) we need to resort to two strong assumptions: Gap Diffie-Hellman [37] and Knowledge of Exponent (KEA1) [14, 18, 4]; in return we get a guarantee that not even the session key computed using the exposed exponent is compromised by this leakage.

We end by noting an important property of our analysis: all results in this paper hold under a strong adversarial model in which the attacker is allowed to register arbitrary public key for all corrupted parties (and at any time during the protocol). This may include a public key that is identical, or related, to the public key of another (possibly uncorrupted) party; in particular, the attacker may not know the private key corresponding to the public key it chose. In practical terms this means that the security of our protocols does not depend on the certification authority requiring registrants of public keys to prove knowledge of the corresponding private keys. This is important since in many practical settings such "proofs of possession" are not required or performed by the

CA (for contrast, see the comparison with [23] in Section 1.6).

1.5 Challenge-Response Signatures

The main technical tool that we develop in order to prove the security of HMQV is a new form of interactive signatures that we call *challenge-response signatures* and which we implement on the basis of a (new) variant of the Schnorr's identification scheme. By applying the Fiat-Shamir methodology to this scheme (and using the proof techniques of Pointcheval-Stern [38]) we obtain challenge-response signatures, named XCR, that are secure in the random oracle model (under the CDH assumption) and have the property that both verifier and signer can compute the *same* signature, the former by knowing the challenge and the latter by knowing the private signature key. The HMQV protocol uses these signatures in an essential way both for the purpose of authentication (of the DH values and the peer identity) as well as for session-key computation. XCR signatures (and their "dual version", called DCR) provide for a natural interpretation of the ideas underlying the MQV design and serve as the basis (technical and conceptual) to the design of HMQV.

In addition, XCR signatures may have applications beyond the HMQV protocol. They do not provide the classical functionality of digital signatures since they are verifier-specific and non-transferable (i.e., cannot be used for non-repudiation purposes). On the other hand, in interactive settings they provide "deniable authentication", an important property for some applications including key exchange. Moreover, for such settings, hashed XCR signatures (denoted HCR) constitute an attractive alternative to DSA signatures. As with DSA, HCR signatures can be implemented over any dlog-based system, including elliptic curves. However, in contrast to DSA, their security is not compromised by the disclosure of ephemeral information (recall that in DSA, the disclosure of a single ephemeral exponent renders the signature scheme totally insecure by revealing the private signing key).

1.6 Related Work.

Implicitly-authenticated DH protocols were first studied in the work of Matsumoto, Takashima and Imai [30] in 1986. Since then this line of research generated many protocols, many of which suffer from various weaknesses. See [9, 34, 7, 8] for some surveys which also include the discussion of desirable security goals for these protocols as well as some of the shortcomings of specific proposals. Two works that study such protocols in a formal model are those of Blake-Wilson et al [8] and Jeong et al [23]. They both treat very similar protocols referred to in the literature as the "unified model". In these protocols, parties $\hat{A}$ and $\hat{B}$ use their public keys g^a, g^b to generate a shared key g^{ab} that they then use to authenticate a DH exchange (c.f., [26]).

The variant studied in [8] is shown to be open to interleaving and known-key attacks and hence insecure (unfortunately, this variant has been widely standardized [2, 3, 20]). One main flaw of this protocol is that it does not explicitly authenticate (or includes under the key derivation hashing) the ephemeral DH values exchanged by the parties. [23] studies the version in which the DH values are included under the key derivation and shows this protocol to be secure in the random oracle model. However, the protocol does not provide resistance to KCI and is open to a UKS attack if the CA does not enforce a proof-of-posession check at time of certificate issuance.[2] Lack of KCI is

[2] In [23], this protocol is also claimed to enjoy perfect forward secrecy (PFS), but what they actually show is a weaker and non-standard notion (see Section 6.2) where PFS holds only for sessions created without active intervention by the attacker. As we pointed out, lack of (full) PFS is an inherent limitation of any 2-message implicitly-authenticated DH exchange, including the 2-message protocols from [23].

one aspect of a more substantial drawback of these protocols, namely, the use of the keys g^{ab} as long-term shared keys between the parties; these keys become particularly vulnerable when cached for efficiency.

In contrast, HMQV is a significantly stronger protocol which, in particular, does not use g^{ab} as a long-term key, does not require (even for efficiency) to cache this value, and even if the value of g^{ab} is ever learned by the attacker it is of no help for impersonating either $\hat{A}$ or $\hat{B}$, or for learning anything about their session keys. On top of all its security advantages (which hold without relying on proofs of possession performed by CAs or prime-order tests performed by the parties), HMQV is more efficient than the unified model protocols that take 3 exponentiations per-party.

Finally, we mention the works of Shoup [41] and Jeong et al [23] that present 2-message authenticated DH exchanges with *explicit* authentication (via signatures and MAC, respectively) that they show to satisfy the security definitions from [41, 6] in the standard (non-random-oracle) model. In these protocols, however, it is sufficient for the attacker to learn a single ephemeral exponent x of a DH value g^x exchanged between parties $\hat{A}$ and $\hat{B}$ to be able to impersonate $\hat{A}$ to $\hat{B}$ indefinitely, and without ever having to learn $\hat{A}$'s or $\hat{B}$'s private keys. This is a serious security weakness which violates the basic principle that the disclosure of ephemeral session-specific information should not compromise other sessions. The reason that these protocols could be proven secure in [41, 23] is that the models of key exchange security considered in these works do not allow the attacker to find any session-specific information beyond the session key itself. The above vulnerability, however, indicates that such models are insufficient to capture some realistic attack scenarios. In contrast, the model of [11], used as the basis for our analysis, captures such attacks via state-reveal queries (see Section 2).

See also Section 3.1 for some more detail on the protocols discussed above, and the concluding remarks in Section 10 for additional comparison to other work.

Organization The paper is organized in the following sections: Section 2 provides a succinct description of the model of secure key-exchange protocols [11] on which all of our analysis work is based (this includes much of the security terminology used in subsequent sections). Section 3 provides background on the MQV protocol as needed for understanding its design, demonstrating its security shortcomings, and motivating the design of HMQV. The presentation in this section is informal and technically "light". Section 4 introduces XCR and DCR signatures which form the basis for our analysis of HMQV (and may be of independent interest), and proves their security (in the random oracle model under the CDH assumption). Section 5 proves the security of HMQV in the formal model of [11] by showing how a successful attacker can be converted into an efficient forger against XCR signatures. Section 6 discusses, defines and proves additional security properties of HMQV not covered by the basic model of [11]. Section 7 is dedicated to analyze the resistance of HMQV to the leakage of ephemeral DH exponents for which it introduces and analyzes "hashed XCR signatures (HCR)". Section 8 presents the 3-message HMQV-C protocol and its added security features. Section 9 presents the one-pass HMQV protocol and its security properties. We end the paper with concluding remarks in Section 10.

2 Security Model for Key-Exchange Protocols

This section presents an abridged description of the Canetti-Krawczyk security model for key-exchange protocols [11] on which all the analysis work in this paper is based. The reader familiar with this model can skip this section. On the other hand, please consult [11] for complete details.

A key-exchange (KE) protocol is run in a network of interconnected parties where each party 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 or to respond 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 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. We will simplify the presentation here by making two assumptions (consistent with the model of [11]): (i) the activation of a session at a party always specifies the name of the intended peer (this is called the "pre-specified peer model" in [12]); and (ii) the session identifier is a quadruple $(\hat{A}, \hat{B}, Out, In)$ where $\hat{A}$ is the identity of the holder of the session, $\hat{B}$ the peer, Out the outgoing messages in the session, and In the incoming messages. In particular, in the case of MQV and HMQV this results in an identifier of the form $(\hat{A}, \hat{B}, X, Y)$ where X is the outgoing DH value and Y the incoming DH value to the session. The peer that sends the first message in a session is called the initiator and the other the responder. We usually denote the peers to a session by $\hat{A}$ and $\hat{B}$; either one may act as initiator or responder. The session $(\hat{B}, \hat{A}, Y, X)$ (if it exists) is said to be matching to the session $(\hat{A}, \hat{B}, X, Y)$. As we'll see, matching sessions play a fundamental role in the definition of security.

In addition, we assume that each party owns a long-term pair of private and public keys, and that other parties can verify the authentic binding between an identity and a public key. For concreteness, we will assume that this binding assurance is provided by a certification authority (CA) which is only trusted to correctly verify the identity of the registrant of a public key before issuing the certificate that binds this identity and the public key. No other actions by the CA are required or assumed; in particular, we make no assumption on whether the CA requires a proof-of-possession of the private key from a registrant of a public key, and we do not assume any specific checks on the value of a public key. In particular, a corrupted party can choose (at any point during the protocol) to register any public key of its choice, including public keys equal or related to keys of other parties in the system. In this paper, a public key will always be a static Diffie-Hellman value and the private key its secret exponent.

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. In particular, *the leakage of session-specific ephemeral secret information should have no adverse effect on the security of any other* non-matching *sessions.*

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 intercept and modify messages sent over these links, it can delay or prevent their delivery, inject its own messages, interleave messages from different sessions, etc. (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 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, for example, the secret exponent of an ephemeral DH value but not the long-term private key used across all sessions at the party). 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 key either via usage of the 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[3].

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 the 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 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 $\mathcal{M}$ among all complete sessions in the protocol, from a random value. When $\mathcal{M}$ chooses the test session it is provided with 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 $\mathcal{M}$ 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 [11] *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, except for a negligible probability, the session key output in these sessions is the same.*

2. *$\mathcal{M}$ 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 KE protocol; in particular, [11] show that a KE protocol that is secure in the above sense is sufficient for the most important application of KE protocols, namely, the establishment of secure communication channels between two parties.

An important property of key-exchange protocols not captured in the above definition is perfect forward secrecy (PFS) [34], 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 [11] via the notion of session-key expiration (which represents the erasure of a session key from memory). A key-exchange protocol is said to be secure with PFS 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.

Additional specific security properties for key-exchange protocols (such as resistance to key-compromise impersonation attacks) are discussed and defined in Sections 3.2 and 6, respectively

[3] The separation between state-reveal and session-key queries from party corruption considers the realistic possibility that private keys be better protected than ephemeral session-specific information.

3 From MQV to HMQV

The MQV protocol was proposed ten years ago and much has been written about it in the literature and in the many standards that adopted it. Yet, relatively little rationale has been given for the specific design choices in the protocol. On the other hand, the authors of MQV [33, 29] have been very explicit about the specific security goals they set for the protocol. In Section 3.1 we attempt to provide some motivation for the MQV design and its underlying ideas (as we understand them). Then, in Section 3.2, we list the stated security goals of MQV and show that the protocol falls short of fulfilling these goals. Finally, in Section 3.3 we show how the lessons learned from the MQV design are applied to the design of HMQV. In particular, while the basic (elegant!) ideas from MQV, as well as the superb performance of the protocol, have been preserved in HMQV, we show how the latter enjoys a much more robust design that provably delivers the security goals envisioned for, but unattained by, the MQV protocol. This whole section is presented informally for the benefit of the non-specialist, leaving the formal treatment and rigorous analysis for later sections.

3.1 Introduction to MQV

A first challenge in designing a 2-message key-exchange protocol is to prevent a successful attack based on the replay of the first protocol message. This is problematic since the first message cannot include any form of a session-specific "freshness guarantee" (such as a nonce or a fresh DH value) contributed by the responder. A solution to this problem is to provide freshness via the computation of the session key. Consider, for example, the following 2-message Diffie-Hellman protocol authenticated using digital signatures and adapted from the ISO protocol [22].

$$\hat{A} \xrightarrow{\quad \hat{A}\,,\ \hat{B}\,,\ g^x\,,\ \textsc{sig}_{\hat{A}}(g^x, \hat{B}) \quad} \hat{B}$$

$$\xleftarrow{\quad \hat{B}\,,\ \hat{A}\,,\ g^y\,,\ \textsc{sig}_{\hat{B}}(g^y, g^x, \hat{A}) \quad}$$

Note that while the inclusion of g^x under the signature of $\hat{B}$ provides freshness to the authentication by $\hat{B}$ this safeguard does not exist in $\hat{A}$'s message. Yet, the session key, g^{xy} is guaranteed to be fresh (and independent from other session keys) as it is randomized by the fresh y. However, also note that the security of the protocol breaks if the attacker is able to find a single pair (x, g^x) used by $\hat{A}$ in a session with $\hat{B}$ (in which case the attacker also learns $\textsc{sig}_{\hat{A}}(g^x, \hat{B})$ from watching the protocol). This allows the attacker to impersonate $\hat{A}$ to $\hat{B}$ indefinitely (using the same message and its knowledge of x) and without ever having to learn $\hat{A}$'s long-term private signature key! This is a serious vulnerability that violates the basic principle that the disclosure of ephemeral session-specific information (e.g., the pair (x, g^x)) should not compromise other sessions. This is particularly serious considering that many applications will compute the pairs (x, g^x) off-line and keep them in less protected storage than the long-term signature key.

Note: As pointed out in Section 1.6, Shoup [41] presents a proof for the above protocol in a security model that does not allow the attacker to find any session-specific information beyond the session key itself. The above vulnerability, however, indicates that such a model is insufficient to capture some realistic attack scenarios. In contrast, the model from [11] used here does capture such attacks via state-reveal queries (see Section 2) and, in particular, allows us to prove that such attacks are not possible against HMQV.

So how can we still design a 2-message protocol such that the authentication in the first message can not be replayed? One solution, put forth in 1986 by Matsumoto, Takashima and Imai [30], is to dispense of the explicit authentication of the DH values and instead provide authentication via the derivation of the session key, namely, by computing this key as a function of the session-specific ephemeral DH values exchanged in the protocol and the long-term private/public keys of the parties (in this setting public keys are static DH keys). In this way, one preserves the original two messages of the basic DH protocol (Figure 1) while thwarting man-in-the-middle attacks via the authenticated key derivation.

A simple instantiation of this idea is to let any pair of parties, $\hat{A}, \hat{B}$, use their respective public keys g^a, g^b in order to compute a shared key g^{ab} which can then be combined with the ephemeral DH key g^{xy} for computing the session key. This approach has been termed the "unified model" in the literature and an insecure instantiation of it (in which the session key is computed as the hash of g^{ab} and g^{xy}) has been extensively standardized [2, 3, 20]. As shown in [8] this protocol falls to known-key and interleaving attacks. A more secure version would also include the ephemeral DH values g^x, g^y under the hash. [23] shows this variant to be secure in the random oracle model provided the CA requires each certificate requester to prove knowledge of the corresponding private key. Clearly, the protocol cannot withstand KCI attacks (see Section 1.2) since knowledge of the private key a of $\hat{A}$ allows the attacker to impersonate any party to $\hat{A}$. This weakness is inevitable when using g^{ab} as a long-term pairwise shared key. Hence, to provide security against KCI attacks (and other forms of secrecy compromise), as well as to avoid the need to trust the CA with proof of possession protocols, we need to depart from the g^{ab} keys.

What we are looking for is a way to mix the values $A = g^a, B = g^b, X = g^x, Y = g^y$ in the key derivation procedure in such a way that as long as the attacker $\mathcal{M}$ does not know both a and x, or both b and y, then $\mathcal{M}$ cannot compute the session key. A natural solution is to compute the session key $K = g^{(x+a)(y+b)}$ (which $\hat{A}$ and $\hat{B}$ compute as $(YB)^{x+a}$ and $(XA)^{y+b}$, respectively). In this case, if an attacker learns x but not a, it cannot compute K. Yet, the protocol is still insecure as the following simple attack shows. $\mathcal{M}$ chooses a value $x^* \in_{\mathrm{R}} Z_q$, computes $X^* = g^{x^*}/A$, and sends X^* to $\hat{B}$ as the initial message from $\hat{A}$. $\hat{B}$ sends $Y = g^y$ and computes the session key $K = (X^*A)^{y+b}$. Unfortunately, also $\mathcal{M}$ can compute K as $(BY)^{x^*}$. Note that if we augment the computation of K to $K = g^{(x+da)(y+eb)}$ for constant d, e the attack is still possible. Moreover, the attack works even if we compute the session key as $H(K)$ for any publicly computable function H (even an ideally random function). On the other hand, if we let d, e vary with X, Y (e.g., in a way that the attacker cannot control e and Y, or d and X, separately) the above simple attack may not work. This idea leads to the the design of MQV where d and e are set to $\bar{X}$ and $\bar{Y}$, respectively (recall that $\bar{X} = 2^\ell + (X \bmod 2^\ell)$ and $\bar{Y} = 2^\ell + (Y \bmod 2^\ell)$ for $\ell = |q|/2$).

But, is this solution secure? We investigates this question next.

3.2 The Insecurity of MQV

As said before, the designers of MQV clearly, albeit informally, stated the security goals behind the MQV design (see [33, 29] and related publications). Here we examine to what extent the MQV protocol achieves these goals in the context of the key-exchange model described in Section 2. We show that some of the most essential security properties, such as resistance to impersonation and known-key attacks, are not satisfied in general by the protocol. We present explicit attacks against most of the properties intended for MQV (in each attack $\mathcal{M}$ succeeds in computing the session key corresponding to an unexposed test session). This stands in strong contrast to HMQV where all these properties are strictly satisfied (see Section 3.3). The presentation here is informal yet some familiarity with the terminology from Section 2 is recommended.

Before going into the specific goals of MQV, let us point to one property that MQV is *not* intended to provide, namely, the assurance to a party completing a session that the peer to the exchange has established the corresponding session key or even that the peer was "alive" at the time of the exchange. Moreover, no 2-message protocol, HMQV included, can provide such assurance (at least not for the responder). However, note that such assurance, while desirable in some applications, is not part of the *core requirements* from a secure key-exchange protocol and, in particular, not ensured by the definition of security from Section 2. What the latter definition ensures is that if a honest party $\hat{A}$ established a session key K with an uncorrupted peer $\hat{B}$ then the only party other than $\hat{A}$ that *may possibly know* the key is $\hat{B}$. This suffices, in particular, to ensure the security of the most important application of key-exchange protocols, namely, secure channels [11]. We expand more on these issues in Section 8 where this property will be added to the protocol at the cost of an extra message.

Resistance to basic impersonation attacks. A minimal requirement for a secure key-exchange protocol is that the attacker, not knowing the private key of a party $\hat{A}$, should not be able to impersonate $\hat{A}$. This minimal requirement, however, cannot be proven to hold in MQV, at least without resorting to specific assumptions on the representation of group elements. To see this consider any group G (of prime order q) and choose a representation of the elements of G such that the $|q|/2$ least significant bits of the representation are fixed. In this case, the MQV's session key becomes $K = H(\sigma)$ where $\sigma = g^{(x+da)(y+eb)}$ and $d = \bar{X}$ and $e = \bar{Y}$ are *fixed constants*. As shown in Section 3.1, in this case the protocol is open to trivial impersonation attacks (the attack is also possible with non-fixed but low-entropy values d and e). In particular, this shows that no analysis of MQV over general prime-order groups can prove the protocol to be secure. Even if one restricts the groups over which an analysis is carried, the following weaknesses of MQV remain.

Resistance to UKS attacks. An unknown-key share (UKS) attack (also known by the more informative terms "source-substitution" [34] or "identity misbinding" [27] attacks) is a particular form of impersonation attack in which the attacker $\mathcal{M}$ interferes with the session establishment between two honest parties $\hat{A}$ and $\hat{B}$ such that at the end of the attack both parties compute the same session key K (which $\mathcal{M}$ may not learn), yet while $\hat{A}$ is convinced that the key is shared with $\hat{B}$, $\hat{B}$ believes that the peer to the session has been $\mathcal{M}$ (here $\mathcal{M}$ can be any corrupted party). This attack, first discovered by Diffie, van Oorschot and Wiener [34], has since become a basic attack under which key-exchange protocols are validated. One can easily see that the MQV protocol is trivially vulnerable to such an attack if $\mathcal{M}$ can register the public key of an honest party $\hat{A}$ under its own (i.e., $\mathcal{M}$'s) identity. It was suggested, however, that the protocol could resist UKS attacks if one disallowed the registration of arbitrary public keys by the attacker; in particular, if the certification authority required a proof of possession by the registrant of a public key (i.e., a proof of knowledge of the corresponding private key). This belief turned out to be wrong as demonstrated by a clever attack due to Kaliski [24] showing that a UKS attack against MQV can be carried by $\mathcal{M}$ using a public key for which $\mathcal{M}$ knows the corresponding private key (in this attack $\mathcal{M}$ chooses its public and private keys after seeing $\hat{A}$'s initial DH value X). A short description of the attack can be found in Appendix A.

For detailed discussions of different aspects of UKS attacks (including its practical consequences) see [16, 24] and many other works in the area. Here we point to one consequence of the attack when seen in light of the model from Section 2. Since the attack forces the same session key, K, to be computed in two *non-matching* sessions, $(\hat{A}, \hat{B}, X, Y)$ and $(\hat{B}, \mathcal{M}, Y, X)$, then the attacker can choose the former as its test session and win the distinguishing game (in other words, the protocol falls to a known-key attack). Interestingly, since the publication of Kaliski's attack it has been believed that one way to solve this weakness of MQV was to move to a 3-message protocol in which MQV is augmented with a "key confirmation" step. Specifically, the protocol is augmented with a MAC value $\text{MAC}_{K_m}(\hat{B}, \hat{A}, Y, X)$ sent in the second message (from $\hat{B}$ to $\hat{A}$) as well as with a third message in which $\hat{A}$ sends $\text{MAC}_{K_m}(\hat{A}, \hat{B}, X, Y)$ (here K_m is derived from $\sigma = \sigma_A = \sigma_B$ while the session key is set to another value derived from σ and "computationally independent" from K_m). The idea is that now the above UKS attack fails since $\hat{B}$ will send $\text{MAC}_{K_m}(\hat{B}, \mathcal{M}, Y, X)$ while $\hat{A}$ expects $\text{MAC}_{K_m}(\hat{B}, \hat{A}, Y, X)$. This, however, is not the case since the attacker can still break the secrecy of the session $(\hat{A}, \hat{B}, X, Y)$. For this, $\mathcal{M}$ breaks into the session $(\hat{B}, \mathcal{M}, Y, X)$ at $\hat{B}$, before $\hat{B}$ sends out the second message, and finds the value of K_m; now $\mathcal{M}$ can compute the MAC value $\text{MAC}_{K_m}(\hat{B}, \hat{A}, Y, X)$ which will convince $\hat{A}$ that she was talking to $\hat{B}$. Here again the attacker can choose the unexposed test session $(\hat{A}, \hat{B}, X, Y)$ and win the distinguishing game.

Perfect Forward Secrecy (PFS) Informally, a key-exchange protocol is said to have the PFS property if the leakage of the long-term key of a party does not compromise the security of session keys established by that party and erased from memory before the leakage occurred. (See Section 2 for a formal definition.) This is another important security property that ensures damage confinement in the case of secrecy leakages, and a main motivation for the use of Diffie-Hellman key-exchange protocols. PFS has also been an explicit design goal for MQV. However, MQV does not achieve it. To see this consider two uncorrupted parties $\hat{A}, \hat{B}$. The attacker $\mathcal{M}$ chooses $x \in_{\mathrm{R}} Z_q$ and sends $X = g^x$ to $\hat{B}$ pretending it to be an initial message for a session coming from $\hat{A}$. Then, $\hat{B}$, acting as responder, chooses $y \in_{\mathrm{R}} Z_q$, sends $Y = g^y$ (supposedly to $\hat{A}$) and completes the session with key $K = H((XA^{\bar{X}})^{y+\bar{Y}b})$. (Note that the matching session will not be established at $\hat{A}$ who did not choose the value X.) Once the session key expires at $\hat{B}$, and is removed from memory, the attacker corrupts $\hat{A}$ and finds the private key a. It now can compute the key K as $H((YB^{\bar{Y}})^{x+\bar{X}a})$ contradicting the PFS property.

We note that this form of attack can be carried against *any* 2-message key-exchange protocol (including HMQV) authenticated via public keys and with no secure shared state previously established between the parties. What can be achieved in a 2-message protocol is a weaker form of forward secrecy in which sessions created without the active involvement of the attacker cannot be recovered after the key expired. While weaker than full PFS this is still an important security feature that we prove to hold in the case of HMQV.

Resistance to Key-Compromise Impersonation (KCI) attacks. Consider the case in which $\mathcal{M}$ has learned the private key a of a party $\hat{A}$. Obviously, $\mathcal{M}$ can now impersonate $\hat{A}$ to any other party. But even in this situation it is desirable to prevent $\mathcal{M}$ from impersonating other (uncorrupted) parties to $\hat{A}$. This property, referred to as *resistance to KCI attacks*, provides the assurance that sessions established by $\hat{A}$ while not being actively controlled by $\mathcal{M}$ remain secure even if her private key is learned by $\mathcal{M}$. This feature has been stated as one of the explicit goals of MQV (in particular, one that differentiates MQV from other implicitly-authenticated KE protocols). In the case of MQV the property states that if $\hat{A}$ establishes a session $s = (\hat{A}, \hat{B}, X, Y)$ where X was chosen by $\hat{A}$ (such that $\mathcal{M}$ does not know x) and $\hat{B}$ is an uncorrupted party then the session s must be secure (i.e., $\mathcal{M}$ cannot distinguish its session key from random without exposing s or its matching session). Since MQV is susceptible to attacks even when the attacker does not know one of the parties' private key then, strictly speaking, it cannot achieve KCI resistance (for a formal treatment of KCI see Section 6). Yet, it is interesting to examine direct KCI attacks against the protocol. We show such an attack in the case that $\mathcal{M}$ has access to the values $\sigma_{\hat{A}}$ or $\sigma_{\hat{B}}$ (remember that the session key is obtained by hashing these values). This demonstrates two important design and specification issues arising from our analysis: (i) The essential contribution of the hashing of the σ values to the security of the protocol (see footnote 1); and (ii) The sensitivity of the protocol to the disclosure of ephemeral state information (e.g., if the computation of σ, which involves the party's private key, is performed in a secure module, then this module should output $H(\sigma)$ rather than σ itself).

Let $\hat{A}$ be a party for whom $\mathcal{M}$ knows the private key a, and $\hat{B}$ be an uncorrupted party. Assume that $\hat{A}$ initiates a session with $\hat{B}$ sending the DH value X (truly chosen by $\hat{A}$ not by $\mathcal{M}$). $\mathcal{M}$ intercepts the message and sends it to $\hat{B}$ but this time with the sender being a party $\hat{A}' \neq \hat{A}$ for which $\mathcal{M}$ knows the private key a' ($\hat{A}'$ can be $\mathcal{M}$ itself or any corrupted party). When $\hat{B}$ responds to $\hat{A}'$ with a DH value Y, $\mathcal{M}$ forwards this message to $\hat{A}$. In this case, $\hat{A}$ and $\hat{B}$ compute $\sigma_{\hat{A}} = (YB^{\bar{Y}})^{x+\bar{X}a}$ and $\sigma_{\hat{B}} = (XA'^{\bar{X}})^{y+\bar{Y}b}$, respectively. Now, if $\mathcal{M}$ learns the value $\sigma_{\hat{B}}$ it can use its knowledge of a and a' to compute $\sigma_{\hat{A}} = \sigma_{\hat{B}} \cdot (YB^{\bar{Y}})^{\bar{X}(a-a')}$, and from it the session key of session $(\hat{A}, \hat{B}, X, Y)$. In other words, the knowledge of a allows $\mathcal{M}$ to break the unexposed session $(\hat{A}, \hat{B}, X, Y)$ (this session is unexposed since the exposed session $(\hat{B}, \hat{A}', Y, X)$ is non-matching) even though $\mathcal{M}$ does not know x and $\hat{B}$ is uncorrupted. This represents a successful KCI attack against MQV. Furthermore, the same attack (which, as said, depends on the ability of the attacker to learn the value of σ) is possible even against the stronger 3-message variant of MQV with key confirmation.

Prime-order testing and exponentiation performance. One potential problem identified by the MQV designers is that since the DH values are not explicitly authenticated by the protocol then it could be possible for an attacker to replace these values. An example of such potential vulnerability is the so called "small group attack" in which the attacker replaces the legitimate DH values of the parties with elements of small order and thus forces the same low-entropy key at both ends. Due to these potential attacks, "implicitly-authenticated" protocols, such as MQV, are often specified to check that the peer's DH value is of the expected prime order q. Unfortunately, this verification adds an exponentiation to the protocol, a drawback particularly noticeable in MQV whose main attractiveness is the virtually optimal performance. To remedy this situation, the MQV designers specified that when the group G is a subgroup of a larger subgroup G', the session key K be computed as $H(\sigma)$ where $\sigma = \sigma_{\hat{A}}^h = \sigma_{\hat{B}}^h$ and h is the cofactor value $|G'|/|G|$. This strategy has two benefits: forcing σ to be in the subgroup G (regardless of whether the DH values were in G or not) and saving the extra exponentiation (the exponentiation to h is for free as it can be combined with the other exponentiations in the protocol). While this specification of MQV certainly ensures that the key is in G, it is not clear whether it solves other potential vulnerabilities. For this reason the standards implementing MQV, as well as various descriptions of the MQV protocol in the literature, often specify (or at least recommend) the performance of the extra exponentiation needed to test that the peer's DH is of prime order q (an operation usually referred to as "ephemeral public key validation").

We do not know if the cofactor exponentiation or an explicitly prime-order test add any real benefit to MQV (in particular, these measures do not prevent any of the attacks shown above). On the other hand, we *prove* that HMQV is secure *without* any one of these operations. This serves to highlight the importance and benefits of an analysis-driven design which frees us from costly precautionary defenses. (All is needed in HMQV is to check that the peer's DH value is non-zero).

Validation of long-term public keys. Both the MQV paper and its standardized version require that the long-term public keys of the parties be validated to be in the prime-order subgroup G. In contrast to the validation of ephemeral DH values as discussed above, in this case the extra computation incurred by this test is easily affordable if it is done by the certification authority at the time of key registration. Yet, since not all CA's will be configured to do such tests, it is better to minimize these extra requirements from CAs. Also in this case, we can prove that the HMQV protocol does not need of such tests. Namely, a dishonest party can choose an element of order other that q as its public key and yet be unable to cause harm to any honest party.

A more significant burden on the CA is to require the latter to check whether the the registrant of a public key knows the corresponding private key. While technically this can be done via zero-knowledge proofs of knowledge, in practice many CAs will not do these checks at all. While originally believed that such "proof of possession" tests could benefit the MQV protocol, this was questioned by Kaliski's UKS attacks [24] (see the discussion above) that are not avoided by these tests. Also here the analysis of HMQV shows that the security of the protocol does not rely on proof-of-possession tests by the CA.

Resistance to disclosure of Diffie-Hellman exponents. As discussed in Section 3.1 an important security requirement from Diffie-Hellman protocols is that the disclosure of the exponent of an ephemeral DH value should not compromise the security of sessions in which that value was not used. In particular, the lack of this property weakens the performance claim that the ephemeral DH exponentiation can be done off-line (an extreme example of such a weakness are DSS signatures where the disclosure of a single ephemeral exponent compromises the long-term signature secret key). While this property has not been considered in the MQV publications, we do investigate it in the setting of HMQV and prove it to hold there. We do not know to what extent it may be satisfied by MQV when the session key is the result of hashing the value σ; without such hashing the property is not achieved (we omit the details of such an attack).

3.3 The security of HMQV

In contrast to MQV, the HMQV protocol presented in this paper can be proven secure in the Canetti-Krawczyk model outlined in Section 2. Moreover, we prove that all the additional security goals of MQV are satisfied by HMQV (with the exception of PFS which cannot be fully achieved by any 2-message protocol). Here we summarize all these properties of HMQV and point out to the design mechanisms that resolve the above weaknesses of MQV. The presentation is still informal and follows the description of properties from the previous subsection; full rigorous proofs are presented in the rest of the paper.

Basic security of HMQV. We prove that HMQV, over any prime order group in which the CDH assumption holds[4], is secure in the sense of Definition 1 when the hash function is modeled as a random oracle. While the proof of this result occupies the largest part of this paper, it is instructive to consider two of the attacks on MQV discussed before as a way to motivate some of the elements in the HMQV design. Specifically, consider the weakness of MQV to group representation and its vulnerability to UKS attacks. Due to our general security proof for HMQV we know that both weaknesses are avoided in HMQV. Specifically, these attacks are prevented via the replacement of the values $d = \bar{X}$ and $e = \bar{Y}$ in MQV with $d = \bar{H}(X, \hat{B})$ and $e = \bar{H}(Y, \hat{A})$. The randomization of X and Y provided by the hashing prevents the weakness of MQV caused by relying on specific group representations, while the identity under the hash binds the peer's identity with the key derivation thus preventing UKS (and other impersonation) attacks.

Resistance to KCI attacks. In Section 6 we formalize KCI attacks in the context of the CK model and show that HMQV provides provable resistance to this attack. For this property we need an additional design element in HMQV, namely, the hashing of $\sigma_{\hat{A}}$ and $\sigma_{\hat{B}}$; without this hashing the

[4] See Section 4.2 for a formalization of the CDH assumption.

following KCI attack is possible. Let $\hat{A}$ be a party for which $\mathcal{M}$ knows the private key a, and $\hat{B}$ be an uncorrupted party. When $\hat{A}$ initiates a session with $\hat{B}$ sending a DH value X chosen by $\hat{A}$ (and whose exponent x the attacker $\mathcal{M}$ does not know), $\mathcal{M}$ replaces X with the DH value X^2. Now $\hat{B}$ computes $K_{\hat{B}} = \sigma_{\hat{B}} = (X^2A^d)^{y+eb}$ (with $d = \bar{H}(X, \hat{B})$, $e = \bar{H}(Y, \hat{A})$). $\mathcal{M}$ issues a session-key query and learns $K_{\hat{B}}$. From this value, and using the knowledge of a, $\mathcal{M}$ computes $(X^2)^{y+eb}$, and by taking a square root $\mathcal{M}$ obtains $(X)^{y+eb}$. Now, using a, $\mathcal{M}$ computes $K_{\hat{A}} = \sigma_{\hat{A}} = (XA^{d'})^{y+eb}$ (here $d' = \bar{H}(X^2, \hat{B})$). Therefore, by exposing the session $(\hat{B}, \hat{A}, Y, X^2)$, the attacker is able to learn the key of the *non-matching* session $(\hat{A}, \hat{B}, X, Y)$. As said, this attack demonstrates the need to define the session key of HMQV to be the hashing of the σ values rather than these values themselves.

Resilience to the disclosure of g^{ab}. The resistance to KCI implies an important characteristic of HMQV that sets it apart from other implicitly-authenticated protocols such as the unified model [8, 23] (see Section 3.1). In particular, in these protocols the value g^{ab} serves as a long-term shared key between parties $\hat{A}$ and $\hat{B}$, and therefore its disclosure suffices for impersonating $\hat{A}$ to $\hat{B}$, and viceversa. In the case of HMQV, and as a corollary of KCI resistance, the disclosure of g^{ab} does not allow impersonation in any direction.

Perfect Forward Secrecy (PFS) As said before, no 2-message protocol (based on public key authentication and without assuming a previous secure exchange between the parties) can provide the PFS property. Thus, this inherent limitation holds for HMQV as well. Yet, HMQV enjoys the highest form of forward secrecy a 2-message protocol can provide: Any session key established by uncorrupted parties *without active intervention* by the attacker is guaranteed to remain secure even if the parties to the exchange are corrupted *after* the session key was erased from the parties memory. This property (which we call 'weak PFS (wPFS)' is formalized and proven in Section 6. Also, as shown in Section 8 the 3-message variant of HMQV augmented with key confirmation provides full PFS.

Prime-order testing and exponentiation performance. HMQV enjoys the best computational performance of any Diffie-Hellman based key exchange protocol authenticated with public keys: 2.5 exponentiations per party. It does not necessitate of any further exponentiations such as the testing for the prime order of the ephemeral DH values required in other protocols and in various standard specifications of MQV. This is a prime example of the effect that a rigorous proof can have on the performance of a protocol by minimizing the safeguards required in its design. Indeed, our analysis of HMQV shows that the only property of the peer's DH value that needs to be verified is that this value is non-zero.

Validation of long-term public keys. Another advantage of the HMQV protocol is that it frees the certification authority from any checks beyond the essential verification of the identity of the registrant of a public key. In particular, the proven security of HMQV does not rely on a registrant of a public key providing proof of possession of the private key to the CA (nor it requires any other checks on the public key).

Resistance to disclosure of Diffie-Hellman exponents. The HMQV protocol can be proven to withstand the leakage of ephemeral DH exponents. Moreover, not only the disclosure of an exponent y of a DH value Y does not compromise sessions in which Y is not used, but even the session in which Y is used is not compromised by the leakage of y (provided, of course, that the private key of the party choosing Y is not known to the attacker). For proving this property (and only for this property) we need to recourse to two stronger assumptions than CDH, namely, the Gap Diffie-Hellman and Knowledge of Exponent assumptions (see Section 7 for the details).

On maximal resistance to exposure attacks. HMQV enjoys a remarkable resistance to disclosure of secret information. Consider a session $(\hat{A}, \hat{B}, X = g^x, Y = g^y)$; the computation of the session key involves the four secret values a, b, x, y. Obviously the disclosure of both a and x, or both b and y, allows the attacker to learn the session key. Remarkably, we prove that the disclosure of any other pair of values in the set $\{a, b, x, y\}$ is insufficient for the attacker to distinguish the session key from randomness. This includes the cases in that the attacker learns both a and b (this is the wPFS property discussed above), both x and y (this follows from the resilience of HMQV to the leakage of ephemeral DH exponents – see Remark 7.1), both a and y, or both b and x (the last two follow from the security of HMQV to KCI attacks). In addition, as we already said, $\hat{A}$ and $\hat{B}$'s communication remains secure even if g^{ab} is learned by the attacker.

HMQV-C: 3-message with key confirmation. While the HMQV protocol provides a full spectrum of basic and advanced security properties it fails to provide two properties that are required or desirable in some applications: full PFS and confirmation from the peer that the session key was computed. The lack of these two properties is unavoidable in HMQV and in any 2-message protocol (without prior communication between the parties). If these properties are to be ensured one can extend HMQV to include a "key confirmation" step. This results in a 3-message protocol (called HMQV-C and described and analyzed in Section 8) that inherits all the security benefits of HMQV plus the above two properties. Another analytical advantage of HMQV-C is that the security proofs for HMQV-C from this paper directly translate into the stronger notion of "universally composable (UC)" security from [13]. See Section 8.

One-pass HMQV. A one-pass key exchange protocol consists of a single message sent from a sender $\hat{A}$ to a recipient $\hat{B}$ from which both parties (using their private and public keys) derive a unique key that only $\hat{A}$ and $\hat{B}$ may possibly know (as long as both parties and the session are uncorrupted). More precisely, the requirements from the established key are the same as in a regular key exchange protocol except for the possibility that the message received by $\hat{B}$ is a replay of an older message from $\hat{A}$. In Section 9 we formalize this notion and show how to build such a protocol from HMQV. This is similar to one-pass MQV but with a proof of security. In addition, this protocol can be used as a secure (in the random oracle model) *authenticated CCA encryption scheme* which is well suited to store-and-forward applications and more efficient than other alternatives.

The rest of the paper is devoted to formalizing and proving all of the above claims about the security of HMQV. We start by introducing and analyzing a main technical tool in our proof: a new type of *challenge-response* signatures based on the Schnorr identification scheme.

4 Exponential Challenge-Response Signatures

Here we introduce the Exponential Challenge-Response (XCR) Signature Scheme which is the main building block used in the design and analysis of the HMQV protocol. As in a regular digital signature scheme, in a challenge-response signature scheme a signer has a pair of private and public keys used for generation and verification, respectively, of signatures. However, in contrast to regular signatures, challenge-response signatures are inherently interactive and require the recipient (i.e., the verifier) of a signature to issue a *challenge* to the signer before the latter can generate the signature on a given message. A *secure* challenge-response signature scheme needs to guarantee that no one other than the legitimate signer be able to generate a signature that will convince the challenger to accept it as valid (in particular, a signature is not only message-specific but also challenge-specific). On the other hand, we are only interested to ensure verifiability of the signature by the challenger, and thus we make no assumptions or requirements regarding the transferability, or verifiability by a third party, of the signature. Moreover, in the scheme described below the party that chooses the challenge can always generate a signature, on any message, which is valid with respect to that particular challenge. What is even more important for our application (and differentiates our scheme from other interactive signatures) is the fact that the verifier can compute, using the challenge, the *same signature string* as the signer.

While the above description may serve as a basis for a general definition of challenge-response signatures, we omit here such a general treatment in favor of a more focused description of the specific challenge-response signature used in this work. In particular, the definition of security is simplified by tailoring it to this specific scheme.

As before, we use g to denote a generator of a group G of prime order q, and $\bar{H}$ to denote a hash function that outputs $\ell = |q|/2$ bits.

4.1 Definition of the XCR Signature Scheme

Definition 2 The exponential challenge-response (XCR) signature scheme. *The signer in a* XCR *scheme, denoted by $\hat{B}$, possesses a private key $b \in_{\mathrm{R}} Z_q$ and a public key $B = g^b$. A verifier (or challenger), denoted $\hat{A}$, provides a message m for signature by $\hat{B}$ together with a challenge X which $\hat{A}$ computes as $X = g^x$ for $x \in_{\mathrm{R}} Z_q$ (x is chosen, and kept secret, by $\hat{A}$). The signature of $\hat{B}$ on m using challenge X is defined as a pair $(Y, X^{y+\bar{H}(Y,m)b})$, where $Y = g^y$ and $y \in_{\mathrm{R}} Z_q$ is chosen by $\hat{B}$. The verifier $\hat{A}$ accepts a signature pair (Y, σ) as* valid *(for message m and with respect to challenge $X = g^x$) if and only if it holds that $Y \neq 0$ and $(YB^{\bar{H}(Y,m)})^x = \sigma$).*

Notation: For a given message m, challenge X, and value Y we define $\text{XSIG}_{\hat{B}}(Y, m, X) \stackrel{\text{def}}{=} X^{y+\bar{H}(Y,m)b}$ (i.e., $\text{XSIG}_{\hat{B}}$ denotes the second element in an XCR *signature pair).*

Relation between XCR and Schnorr's scheme. The main motivation for introducing the XCR scheme comes from its use in our design and analysis of the HMQV protocol (Section 5). By now, however, it may be illustrative to motivate this scheme via its relation to the Schnorr's identification scheme from which the XCR scheme is derived. We sketch this relation next. Schnorr's (interactive) identification scheme consists of a proof of knowledge of the discrete logarithm b for a given input $B = g^b$. Let $\hat{B}$ denote the prover in this scheme (that possesses b) and $\hat{A}$ the verifier (that is given the input B). The basic Schnorr's identification consists of three messages: (i) $\hat{B}$ chooses $y \in_{\mathrm{R}} Z_q$ and sends $Y = g^y$ to $\hat{A}$; (ii) $\hat{A}$ responds with a random value $e \in_{\mathrm{R}} Z_q$; and (iii) $\hat{B}$ sends $\hat{A}$ the value $s = y + eb$. $\hat{A}$ accepts if and only if $g^s = YB^e$ holds.
This protocol is a Arthur-Merlin zero-knowledge proof of knowledge (of b) for an honest verifier $\hat{A}$ (i.e., one that chooses e uniformly at random). Therefore, it can be transformed via the Fiat-Shamir methodology into a signature scheme, namely $\text{SIG}_{\hat{B}}(m) = (Y, y + \bar{H}(Y,m)b)$, that is provably secure in the random oracle model [38].

Now consider the following 4-message variant of Schnorr's protocol in which a first message from $\hat{A}$ to $\hat{B}$ is added. In this first message $\hat{A}$ sends to $\hat{B}$ a value $X = g^x$. Then the 3 messages from Schnorr's scheme follow, except that in message (iii) (the fourth message in the modified protocol) rather than sending $s = y + eb$ to $\hat{A}$, $\hat{B}$ sends $S = X^s$. $\hat{A}$ accepts if and only if $S = (YB^e)^x$. It can be shown that this protocol is a proof of the "ability" of $\hat{B}$ to compute $CDH(B, X)$ for any value $X \in G$. Moreover, the protocol is zero-knowledge against a verifier $\hat{A}$ that chooses e at random (while X may be chosen arbitrarily). Now, note that applying the Fiat-Shamir transformation to this protocol one obtains the challenge-response signature XCR.[5] This also explains why we use

[5] Note that if in Schnorr's protocol one chooses e in $\{0,1\}^\ell$ (rather than from Z_q) the protocol remains valid except

the term "exponential" in naming the XCR scheme: it refers to the replacement of $s = y + eb$ in the Schnorr scheme with X^s in the last message of the protocol.

Next we establish our security requirement from the XCR scheme.

Definition 3 Security of the XCR signature scheme. *We say that the* XCR *challenge-response signature scheme is secure if no polynomial-time machine $\mathcal{F}$ can win the game in Figure 3 with non-negligible probability.*

Forger $\mathcal{F}$ in Definition 3

1. $\mathcal{F}$ is given values B, X_0 where $B, X_0 \in_{\mathrm{R}} G$.
2. $\mathcal{F}$ is given access to a signing oracle $\hat{B}$ (representing a signer $\hat{B}$ with private key b and public key B) which on query (X, m) outputs a pair $(Y, \mathit{XSIG}_{\hat{B}}(Y, m, X))$ where $Y = g^y, y \in_{\mathrm{R}} Z_q$, is chosen by $\hat{B}$ afresh with each query.
3. $\mathcal{F}$ is allowed a polynomial number of queries to $\hat{B}$ where the queries are chosen (possibly adaptively) by $\mathcal{F}$.
4. $\mathcal{F}$ halts with output "fail" or with a *guess* in the form of a triple (Y_0, m_0, σ).

$\mathcal{F}$'s guess is called a (successful) *forgery* if the following two conditions hold:

(a) The pair (Y_0, σ) is a valid XCR signature of $\hat{B}$ on message m_0 with respect to challenge X_0 (i.e., $Y_0 \neq 0$ and $\sigma = \mathit{XSIG}_{\hat{B}}(Y_0, m_0, X_0)$; also note that the value of X_0 is the one received by $\mathcal{F}$ as input).

(b) The pair (Y_0, m_0) did not appear in any of the responses of $\hat{B}$ to $\mathcal{F}$'s queries.

We say that $\mathcal{F}$ wins the game (or simply *forges*) if it outputs a successful forgery.

Figure 3: Forgery Game for XCR Signatures

Remarks on Definition 3.

1. Note that in order to be successful the forger has to use the input X_0 in its forgery. This captures the fact that XCR signatures are only unforgeable with respect to a challenge not chosen by the attacker (indeed, a signature with a given challenge can be trivially computed by the party choosing the challenge, i.e. the one that knows the dlog of the challenge).
2. According to our definition, if $\hat{B}$ outputs a signature (Y, σ) on a message m wrt challenge X, and the forger can find a signature (Y', σ') for the same m wrt the same challenge X but with $Y' \neq Y$, then we consider $\mathcal{F}$ successful (namely, finding a second signature for the same message is considered a valid forgery). In some sense, we look at these signatures as signing both m and Y. This property may not be essential in other applications of challenge-response signatures but it is crucial for the application to HMQV security in this paper.

that the soundness is limited by $2^{-\ell}$. This is the basis for our choice of $\ell = |q|/2$ as the output length of $\bar{H}(Y, m)$, namely, a trade-off between efficiency and security. See Remark 4.2 for a more accurate discussion.

3. We do not ask that $\mathcal{F}$ will always output good forgeries; it can output "fail" or even invalid triples. The only requirement is that with non-negligible probability (over the distribution of inputs to $\mathcal{F}$, the choices by the random oracle $\bar{H}$, the coins of $\mathcal{F}$, and the coins of $\hat{B}$) $\mathcal{F}$ output a successful forgery.

4. Note that we only restricted Y_0 to be non-zero. In particular, we require no check that Y_0 be of prime order q (only that it represents a non-zero element for which the group operation is defined). This is an important aspect of XCR signatures. In particular, the requirement to run a prime-order test would translate into an additional exponentiation for each party in the HMQV protocol, thus degrading significantly the "almost optimal" performance of the protocol.

4.2 Proof of Unforgeability for the XCR Signature Scheme

The security of XCR signatures (and of all other results in this paper) depends on the Computational Diffie-Hellman (CDH) Assumption that we define below. We first need the following remark (or disclaimer) on our treatment of computational issues.

Remark 4.1 (Asymptotics.) The technical presentation of computational results in this paper uses the probabilistic polynomial-time abstraction (leaving a detailed concrete security analysis for further work). The implicit security parameter used throughout the paper is the quantity $|q|$ (where q denotes the order of the group G). As customary, we will use the term negligible to denote a function on the security parameter that decreases faster than any inverse polynomial fraction while non-negligible denotes a function on the security parameter that is lower bounded by an inverse polynomial fraction. (Note that "negligible" and "non-negligible" are not the complement of each other.) Here and in the sequel, our treatment of asymptotics is informal. For example, we state claims in terms of single groups while a formal statement should consider infinite families of finite groups. We believe that these details are relatively straightforward to fill in and thus we prefer to sacrifice formal accuracy for easier understanding.

Definition 4 *For two elements $U = g^u, V = g^v$ in G we denote by $CDH_g(U, V)$ the result of applying the Diffie-Hellman computation (wrt to generator g) to U and V, i.e., $CDH_g(U, V) = g^{uv}$. (For simplicity we will often omit the subscript g from the CDH_g notation.) An algorithm is called a* CDH solver for *G if it takes as input pairs of elements (U, V) in G and a generator g of G and outputs the Diffie-Hellman result $CDH_g(U, V)$.*
We say that the Computational Diffie-Hellman (CDH) *assumption holds in the group $G = \langle g \rangle$ if for all polynomial-time CDH solvers for G, the probability that on a pair (U, V), for $U, V \in_R G$, the solver computes the correct value $CDH_g(U, V)$ is negligible (the probability taken over the random coins of the solver and the choice of U, V independently at random in G).*

The following theorem states the security of the XCR scheme in the random oracle model under the CDH assumption; it constitutes the basis for the proof of security of protocol HMQV.

Theorem 5 *Under the CDH assumption, the* XCR *signature scheme is secure (according to Definition 3) in the random oracle model.*

Proof: Given an efficient and successful forger $\mathcal{F}$ against the XCR signature scheme (i.e., $\mathcal{F}$ wins the forgery game from Definition 3 with non-negligible probability), we build an efficient solver $\mathcal{C}$

for the CDH problem, namely, $\mathcal{C}$ gets as input a pair of random elements U, V in G, and outputs the value $CDH(U, V)$ with non-negligible probability. Unsuccessful runs of $\mathcal{C}$ may end with "fail" or just the wrong value of $CDH(U, V)$. Using results by Maurer and Wolf [31] and Shoup [40] such a "faulty CDH solver" can be transformed, using the self-reducibility properties of the CDH problem, into an efficient algorithm that solves CDH for every input U, V with only negligible probability of error.

Algorithm $\mathcal{C}$ is presented in Figure 4, and it follows a mostly standard argument for Fiat-Shamir type signatures. The idea is that if $\mathcal{F}$ can succeed in forging a signature with a pair (Y_0, m_0) and a given value $\bar{H}(Y_0, m_0)$ output by the function $\bar{H}$, then $\mathcal{F}$ is likely to succeed also when $\bar{H}(Y_0, m_0)$ is set to a *different* random value. Using this property, we construct $\mathcal{C}$ such that after running $\mathcal{F}$ twice, $\mathcal{C}$ obtains (with non-negligible probability) two forgeries with the same pair (Y_0, m_0) but different values of $\bar{H}(Y_0, m_0)$. Now, using these two forgeries $\mathcal{C}$ is able to compute $CDH(U, V)$.

Examining the specification of $\mathcal{C}$ in more detail, first note that in the run of $\mathcal{F}$ by $\mathcal{C}$ all queries to the signer $\hat{B}$ are answered by $\mathcal{C}$ without knowledge of the private key b, and without access to an actual signing oracle for $\hat{B}$. Instead all these answers are simulated by $\mathcal{C}$ in steps S1-S3. It is easy to see that this is a perfect simulation of the XCR signature generation algorithm under private key b except for the following deviation that happens with negligible probability: In step S3 of the simulation, $\mathcal{C}$ does not complete the run of $\mathcal{F}$ if the value (Y, m) was queried earlier from $\bar{H}$. However, since the value Y, as generated by $\mathcal{C}$, is distributed uniformly over G and chosen independently of previous values in the protocol, then the probability that the point (Y, m) was queried earlier from $\bar{H}$ is at most Q/q where Q is an upper bound on the number of queries to $\bar{H}$ that occur in a run of $\mathcal{C}$.

Therefore, the probability of $\mathcal{F}$ outputting a successful forgery in the run under $\mathcal{C}$ is the same, up to a negligible difference, as in a real run of $\mathcal{F}$, and therefore non-negligible. In particular, when such a successful forgery is output by $\mathcal{F}$ then conditions F1 and F2 checked by $\mathcal{C}$ necessarily hold. Condition F3 also holds except for probability $2^{-\ell}$, i.e., the probability that $\mathcal{F}$'s forgery is correct when it did not query $\bar{H}(Y_0, m_0)$. To see this, note that if one fixes the pair (Y_0, m_0) and the challenge X_0, then the signature produced with $e = \bar{H}(Y_0, m_0)$ is necessarily different than the signature produced with $e' = \bar{H}(Y_0, m_0)$ if $e \neq e' \pmod{q}$. Hence, the probability that $\mathcal{F}$ will guess the right signature without querying $\bar{H}(Y_0, m_0)$ is at most as the probability of guessing the value of $\bar{H}(Y_0, m_0)$, i.e., $2^{-\ell}$. Since conditions F1-F3 determine the run of the "repeat experiment" then in all we have that the simultaneous probability that $\mathcal{F}$ outputs a correct forgery in a run under $\mathcal{C}$ AND that $\mathcal{C}$ executes the "repeat experiment" in that run is non-negligible.

Now, using the Forking Lemma[6] from [38] we obtain that the probability that in the "repeat experiment" $\mathcal{F}$ will output a correct forgery for the pair (Y_0, m_0) given that $\mathcal{F}$ did so in the first run is non-negligible. Moreover, in such a case we are guaranteed that the forgeries in the first and second runs use *different* values e, e' of $\bar{H}(Y_0, m_0)$.

We now proceed to show that in the case that both the first run and the repeat experiment end up with two valid forgeries for the same pair (Y_0, m_0), and $e \neq e'$ (which happens with probability $1 - 2^{-\ell}$), then the value W computed by $\mathcal{C}$ equals $CDH(X_0, B)$. Indeed, a simple computation

[6] We note that the formulation of the Forking Lemma in [38] is very general and fits exactly our application: namely, it applies to signatures of the form $(m, Y, \bar{H}(Y, m), \sigma)$ derived via the Fiat-Shamir methodology, provided that the signer is simulatable without knowledge of its private key (the use of the challenge X in our case does not affect the validity of the Forking Lemma argument).

Building a CDH solver $\mathcal{C}$ from an XCR forger $\mathcal{F}$

Setup. Given a successful XCR-forger $\mathcal{F}$ we build an algorithm $\mathcal{C}$ to solve the CDH problem. The inputs to $\mathcal{C}$ are random values $U = g^u, V = g^v$ in G, and its goal is to compute $CDH(U, V) = g^{uv}$.

$\mathcal{C}$'s actions. $\mathcal{C}$ sets $B = V$ and $X_0 = U$, and runs the forger $\mathcal{F}$ on input (B, X_0) against a signer $\hat{B}$ with public key B. $\mathcal{C}$ provides $\mathcal{F}$ with a random tape and provides random answers to the $\bar{H}$ queries generated in the run (if the same $\bar{H}$ query is presented more than once $\mathcal{C}$ answers it with the same response as in the first time).

Each time $\mathcal{F}$ queries $\hat{B}$ for a signature on values (X, m) chosen by $\mathcal{F}$, $\mathcal{C}$ answers the query for $\hat{B}$ as follows (note that $\mathcal{C}$ does not know b):

S1. Chooses $s \in_R Z_q, e \in_R \{0,1\}^\ell$.

S2. Sets $Y = g^s/B^e$.

S3. Sets $\bar{H}(Y, m) = e$ (if $\bar{H}(Y, m)$ was defined by a previous query to $\bar{H}$, $\mathcal{C}$ aborts its run and outputs "fail").

$\mathcal{C}$ responds to $\mathcal{F}$'s query with the signature pair (Y, X^s)

When $\mathcal{F}$ halts $\mathcal{C}$ checks whether the three following conditions hold:

F1. $\mathcal{F}$ output a guess (Y_0, m_0, σ), $Y_0 \neq 0$.

F2. The pair (Y_0, m_0) was not used as the (Y, m) pair in any of the signatures generated by $\hat{B}$.

F3. The value $\bar{H}(Y_0, m_0)$ was queried from the random oracle $\bar{H}$.

If the three conditions hold, then $\mathcal{C}$ proceeds to the "repeat experiment" below; in all other cases $\mathcal{C}$ halts and outputs "fail".

The repeat experiment. $\mathcal{C}$ runs $\mathcal{F}$ again for a second time under the same input (B, X_0) and using the same coins for both $\mathcal{C}$ and $\mathcal{F}$. The difference between the two runs is in the way in which $\mathcal{C}$ answers the $\bar{H}$ queries during the second run. Specifically, all queries to $\bar{H}$ performed before the $\bar{H}(Y_0, m_0)$ query are answered identically as in the first run. The query $\bar{H}(Y_0, m_0)$, however, is answered with a new independent value $e' \in_R \{0,1\}^\ell$. Subsequent queries to $\bar{H}$ are also answered at random from $\{0,1\}^\ell$, independently of the responses provided in the first run.

Output. If at the end of the second run, $\mathcal{F}$ outputs a guess (Y_0, m_0, σ') (with same (Y_0, m_0) as in the first run) and $e \neq e'$, then $\mathcal{C}$ computes the value $W = (\sigma/\sigma')^{(e-e')^{-1}}$ and outputs W as its guess for $CDH(U, V)$; otherwise $\mathcal{C}$ outputs "fail".

Figure 4: Proof of Theorem 5: Reduction from CDH to XCR forgeries

shows that, if $Y_0 \neq 0$ as necessary for a valid forgery, then writing $X_0 = g^{x_0}$ we have:

$$W = \left(\frac{\sigma}{\sigma'}\right)^{\frac{1}{e-e'}} = \left(\frac{(YB^e)^{x_0}}{(YB^{e'})^{x_0}}\right)^{\frac{1}{e-e'}} = (B^{(e-e')x_0})^{\frac{1}{e-e'}} = B^{x_0} = CDH(X_0, B).$$

Now, since X_0 and B are, respectively, the inputs U and V provided to $\mathcal{C}$, then we get that in this case (which happens with non-negligible probability) $\mathcal{C}$ has successfully computed $CDH(U, V)$.

□

Remark 4.2 (Number of bits in $\bar{H}(Y, m)$). Let ℓ be the number of bits in the output of $\bar{H}(Y, m)$. Clearly, the smaller ℓ the more efficient the signature scheme is; on the other hand, a too small ℓ implies a bad security bound (since once the exponent $\bar{H}(Y,m)$ is predictable the signature scheme is insecure). But how large a ℓ do we need for security purposes? Here we see that setting $\ell = \frac{1}{2}|q|$, as we specified for XCR signatures (and for its application to the HMQV protocol), provides the right performance-security trade-off. In order to assess the level of security that ℓ provides to the XCR signatures (and consequently to HMQV), we note that there are two places in the above proof where this parameter ℓ enters the analysis. One is when bounding the probability that the attacker could forge a signature with parameters (Y_0, m_0) without querying $\bar{H}$ on this pair. As we claimed the probability in this case is $2^{-\ell}$ (or $1/\sqrt{q}$ using the fact that we defined $\ell = |q|/2$). The other use of ℓ is in the proof of the Forking Lemma by Pointcheval and Stern [38]. When written in terms of XCR signatures Lemma 9 and Theorem 10 from [38] show that given a forger against XCR signatures that works time T, performs Q queries to $\bar{H}$ and forges with probability ε, one can build a CDH solver that runs expected time $c\frac{Q}{\varepsilon}T$ provided that $\varepsilon \geq \frac{c'Q}{2^\ell}$ (c, c' are constants). Now, since we know how to build CDH solvers that run time $\sqrt{q}$ (e.g., Shanks algorithm) then the above analysis tells us something significant only when $c\frac{Q}{\varepsilon}T << \sqrt{q}$, in particular $Q/\varepsilon << \sqrt{q}$. From this and the condition $\varepsilon \geq c'Q/2^\ell$ we get that we need $Q/\varepsilon < \min\{\sqrt{q}, 2^\ell\}$. Since this is the only constraint on ℓ we see that choosing ℓ such that $2^\ell > \sqrt{q}$ does not add to the security of the scheme, and therefore setting $\ell = \frac{1}{2}|q|$ provides the best trade-off between security and performance. (Note that the same length consideration applies to the parameter e in the modified Schnorr's identification scheme described following Definition 2). We also comment, independently from the above considerations on ℓ, that the above constraint $Q < \varepsilon\sqrt{q}$ also guarantees that the simulation error in step S3 of Figure 4 (which we showed in the proof of Theorem 5 to be at most Q/q) is no more than $1/\sqrt{q}$.

Remark 4.3 (Changing the order of interaction with $\hat{B}$.) In our application of XCR signatures to the analysis of the HMQV protocol we change the order of interaction between the challenger $\hat{A}$ and the signer $\hat{B}$. In Definition 2, $\hat{A}$ presents $\hat{B}$ with the message m at the same time that it provides the challenge X to $\hat{B}$, thus allowing $\hat{B}$ to immediately respond with the signature pair $(Y, \mathit{XSIG}_{\hat{B}}(Y,m,X))$. In the modified version that we will consider, we have the following order of interaction: (i) $\hat{A}$ presents message m to $\hat{B}$ and $\hat{B}$ outputs Y, then (at some later point) (ii) $\hat{A}$ provides (Y,m,X) to $\hat{B}$, and $\hat{B}$ outputs $\mathit{XSIG}_{\hat{B}}(Y,m,X)$. Now, we can modify the forging game of Definition 3 to allow $\mathcal{F}$'s queries to $\hat{B}$ to take this modified order. In particular, $\mathcal{F}$ can interleave different interaction with $\hat{B}$, namely, it can run several instances of step (i) before running the corresponding steps (ii). This requires $\hat{B}$ to keep state after step (i) with the values of Y, y and m. When $\mathcal{F}$ later presents (Y,m,X) in step (ii), $\hat{B}$ checks that it has the pair (Y,m) in its state and if so it responds with $\mathit{XSIG}_{\hat{B}}(Y,m,X)$ and erases (Y,m) from its state (if $\hat{B}$ did not have the pair (Y,m) in its state then it does not issue the signature). Note that this specification of $\hat{B}$'s actions ensures that $\hat{B}$ will never use the same value of Y for two different signatures. It can be easily verified that the above proof of security of XCR signatures remains valid for this modified order (simply because the simulation of the choice of Y by $\hat{B}$ does not require the knowledge of X, but only the value of m needed to determine $\bar{H}(Y,m)$).

Remark 4.4 (A non-interactive XCR **variant.)** XCR signatures can be made non-interactive, but verifier-specific, by putting $X = A$, where A is a public key of the verifier. In this case the signature will be a pair (Y,t) where t is a MAC tag computed on the signed message using a key derived by hashing $\mathit{XSIG}_{\hat{A}}(Y, \text{“}\hat{A}\text{”}, A)$. This provides for a very efficient non-interactive verifier-specific deniable authentication mechanism (see Section 9). It does *not* provide for a universally-verifiable non-repudiable signature.

Remark 4.5 (HCR and DSS signatures.) We do not know whether XCR signatures remain secure if the exponent y corresponding to a value Y used by $\hat{B}$ in a signature is revealed to the forger (note that in this case the simulation steps S1-S3 in Figure 4 do not work). On the other hand, if one modifies the definition of XCR such that the $\mathit{XSIG}_{\hat{B}}$ component is replaced with a hash of this value (note that the signature is still verifiable by the challenger) then one obtains a signature scheme in which revealing y does not help the forger. More precisely, in Section 7 we study these signatures, which we call HCR, in detail and show that under the Gap Diffie-Hellman and KEA1 assumptions they are unforgeable in the random oracle model even if y is revealed to the attacker. As a result, HCR signatures provide for a more secure alternative to DSS signatures as they resolve the main DSS vulnerability by which the disclosure of a single ephemeral exponent (i.e., k in the component $r = g^k$ of a DSS signature) suffices to reveal the signature key. On the other hand, HCR signatures are verifier-specific and require interaction (or the possession of a public key by the verifier as in Remark 4.4), and do not provide for third-party verifiability (a property that may be a bug or a feature of HCR depending on the application).

4.3 Dual XCR Signatures (DCR)

An important property of XCR signatures is that the challenger (having chosen the challenge) can compute the signature by itself. Here we show how to take advantage of this property in order to derive a related challenge-response signature scheme (which we call the "dual XCR scheme", or DCR for short) with the property that any two parties, $\hat{A}, \hat{B}$, can interact with each other with the dual roles of challenger and signer, and each produce a signature that no third party can forge. Moreover, *and this is what makes the scheme essential to the HMQV protocol,* the resultant signatures by $\hat{A}$ and by $\hat{B}$ have the *same value.* (More precisely, they have the same XSIG component.)

Definition 6 The dual (exponential) challenge-response (DCR) signature scheme. *Let $\hat{A}, \hat{B}$ be two parties with public keys $A = g^a, B = g^b$, respectively. Let m_1, m_2 be two messages. The* dual XCR signature (DCR for short) *of $\hat{A}$ and $\hat{B}$ on messages m_1, m_2, respectively, is defined as a triple of values: X, Y and $\mathit{DSIG}_{\hat{A},\hat{B}}(m_1, m_2, X, Y) \overset{\text{def}}{=} g^{(x+da)(y+eb)}$, where $X = g^x, Y = g^y$ are challenges chosen by $\hat{A}$ and $\hat{B}$, respectively, and the symbols d and e denote $\bar{H}(X, m_1)$ and $\bar{H}(Y, m_2)$, respectively.*

As said, a fundamental property of a DCR signature is that after exchanging the values X and Y (with x and y chosen by $\hat{A}$ and $\hat{B}$, respectively), both $\hat{A}$ and $\hat{B}$ can compute (and verify) the **same** signature $\mathit{DSIG}_{\hat{A},\hat{B}}(m_1, m_2, X, Y)$. This can be seen from the identities:

$$\mathit{DSIG}_{\hat{A},\hat{B}}(m_1, m_2, X, Y) = g^{(x+da)(y+eb)} = (YB^e)^{x+da} = (XA^d)^{y+eb}$$

Moreover, as shown next the attacker cannot feasibly compute this signature.

The security of DCR signatures. Roughly speaking, a dual signature is an XCR signature by $\hat{A}$ on message m_1, under challenge YB^e, and at the same time an XCR signature by $\hat{B}$ on message m_2, under challenge XA^d. More precisely, since the values d and e are determined during the signature process (via the possibly adversarial choice of messages m_1, m_2), then we will say that a DCR signature of $\hat{B}$ is secure (with respect to A) if no efficient attacker can win, with non-negligible probability, the game of Figure 3 with the following modifications. In step 2, the queries to $\hat{B}$ are of the form (X, m, m_1) and the signature by $\hat{B}$ is the pair $(Y, \mathit{XSIG}_{\hat{B}}(Y, m, XA^d))$ where

Y is chosen by $\hat{B}$ and $d = \bar{H}(X, m_1)$. A successful forgery is a quadruple (Y_0, m_0, m_1, σ) where $\sigma = \mathit{XSIG}_{\hat{B}}(Y_0, m_0, X_0 A^d)$, $Y_0 \neq 0$, the pair (Y_0, m_0) satisfies the validity requirement (b) from Figure 3, and m_1 is an arbitrary message chosen by $\mathcal{F}$. We say that the dual signature of $\hat{B}$ is secure if it is secure *with respect to any value* $A = g^a$ *not chosen by the attacker.*

Theorem 7 *Let $\hat{A}, \hat{B}$ be two parties with public keys $A = g^a, B = g^b$, respectively. Under the CDH assumption, the* DCR *signature of $\hat{B}$ with respect to A is secure even if the forger is given the private key a of $\hat{A}$ (but not the private key of $\hat{B}$).*

Proof: The same proof of unforgeability of XCR (Theorem 5) works here with a modified computation of W as specified below. First note that since the DCR signature of $\hat{B}$ now involves the value $d = \bar{H}(X_0, m_1)$, where m_1 is a message that $\mathcal{F}$ may choose at will, then the value of m_1 chosen by $\mathcal{F}$ before the repeat experiment may differ from the value of m_1 chosen during the repeat experiment. In this case we get two different values, d, d', used in the σ and σ' signatures. Also, note that even with a single value of d the specification of W in the proof of Theorem 5 would result in the value $(XA^d)^b$ rather than X^b (as required in order to solve the CDH problem on inputs $U = X, V = B$). We deal with these two issues by redefining W as follows (here we use the fact that a, the private key of $\hat{A}$, is known to $\mathcal{C}$):

$$W = \left(\frac{\sigma/(YB^e)^{da}}{\sigma'/(YB^{e'})^{d'a}} \right)^{\frac{1}{e-e'}} \tag{1}$$

The rest of the proof remains unchanged. □

Remark 4.6 The above Theorem assumes that the value b is not known while a is known (to the attacker and to the CDH solver $\mathcal{C}$). While this results in a strong assurance of security (namely, the DCR signatures of $\hat{B}$ are secure even if the private key of the peer, $\hat{A}$, is given to the attacker), it also implicitly assumes that $\hat{A}$ and $\hat{B}$ are different entities or, more precisely, that $A \neq B$. Indeed, when $\hat{A} = \hat{B}$ both a and b are secret (they are the same value) and therefore the computation of the CDH value W cannot be carried out as specified in the above proof using a. For the time being we will assume $A \neq B$ and use the above theorem. Only in Section 6.3 we will return to the issue of $A = B$ when dealing with *reflection attacks* against the HMQV protocol. There we will see that we can extend the Theorem to the case $A = B$, at least with some restrictions on the forger $\mathcal{F}$ that are satisfied in the HMQV setting.

HMQV in a nutshell. The HMQV protocol consists of an exchange between parties $\hat{A}$ and $\hat{B}$ of DH values $X = g^x$ and $Y = g^y$ that serve as challenges from which both parties compute the dual XCR signature $\mathit{DSIG}_{\hat{A},\hat{B}}(``\hat{A}", ``\hat{B}", X, Y) = g^{(x+da)(y+eb)}$. The session key is then derived by hashing this value. In this way the signature itself need not be transmitted: it is the uniqueness of the signature that ensures a common derived value for the session key, and it is the *ability to compute the key (equivalently, the signature)* that provides for a proof that the exchange was carried by the alleged parties $\hat{A}$ and $\hat{B}$. Moreover, since the messages m_1, m_2 on which the signature is computed are the identities of the peers, both parties get assurance that the key they computed is uniquely bound to the correct identities (this is essential to avoid some authentication failures such as the UKS attacks). While the casting of the HMQV design in terms of DCR signatures is the main conceptual contribution of our work, showing that this idea indeed works for proving the security of the protocol turns out to be technically involved. The rest of the paper is devoted to this proof.

5 Basic Security Analysis of the HMQV Protocol

Using the terminology from the previous section, a session of the HMQV protocol between two parties $\hat{A}, \hat{B}$ consists of a basic Diffie-Hellman exchange of DH values $X = g^x$ and $Y = g^y$ (Figure 1) with the session key computed as $H(\pi)$ where $\pi = \textit{DSIG}_{\hat{A},\hat{B}}(m_1 = \hat{B}, m_2 = \hat{A}, X, Y)$. That is, π is computed as the dual signature of $\hat{A}$ and $\hat{B}$ on each other's identity. In the sequel, we denote the above signature by the shorthand $\pi(\hat{A}, \hat{B}, X, Y)$, namely,

$$\pi(\hat{A}, \hat{B}, X, Y) \stackrel{\text{def}}{=} \textit{DSIG}_{\hat{A},\hat{B}}(m_1 = \hat{B}, m_2 = \hat{A}, X, Y) = g^{(x+da)(y+eb)}$$

where $d = \bar{H}(X, \hat{B})$, $e = \bar{H}(Y, \hat{A})$, and $A = g^a, B = g^b$ are the public keys of parties $\hat{A}, \hat{B}$ respectively. Note that $\pi(\hat{A}, \hat{B}, X, Y) = \pi(\hat{B}, \hat{A}, Y, X)$.

In this section we prove that the HMQV protocol is secure in the Canetti-Krawczyk model of key exchange [11] outlined in Section 2; additional security properties of HMQV are proven in Section 6. We start with a full specification of the HMQV protocol, in accordance to the above formal model of key exchange.

5.1 Detailed Specification of the HMQV Protocol

In our formal treatment we will refer to the HMQV protocol as protocol Π.

Protocol Π runs in a multi-party network where any of the parties can be invoked to run the protocol. Each invocation of the protocol at a party creates a *session* (a local state containing information related to this specific instance of the protocol), which may produce outgoing messages and the output of a session key at completion. During a session, a party can be activated with three types of activations presented next (in the following description $\hat{A}$ denotes the identity of the party being activated and $\hat{B}$ the identity of of the intended peer to the session).

Session Activations

Initiate$(\hat{A}, \hat{B})$*:* $\hat{A}$ *generates a value* $X = g^x, x \in_{\mathrm{R}} Z_q$*, creates a local session of protocol* Π *which she identifies as (the incomplete) session* $(\hat{A}, \hat{B}, X)$*, and outputs the value* X *as its outgoing message.*
The meaning of this activation is that $\hat{A}$ has been activated as the initiator of a session with $\hat{B}$, and X is the message intended to be delivered to peer $\hat{B}$ as part of this session. We call $\hat{A}$ the holder (or owner) of the session, $\hat{B}$ the peer to the session, and X the outgoing (DH) value.

Respond$(\hat{A}, \hat{B}, Y)$*:* $\hat{A}$ *checks that* $Y \neq 0$*, if so it generates a value* $X = g^x, x \in_{\mathrm{R}} Z_q$*, outputs* X*, and completes a session with identifier* $(\hat{A}, \hat{B}, X, Y)$ *and session key* $H(\pi(\hat{A}, \hat{B}, X, Y))$*.*
Here $\hat{A}$ is being activated as the responder in the session with peer $\hat{B}$ and incoming value Y. In this case, $\hat{A}$ immediately completes its session (there are no further incoming messages). Note that if the incoming value Y is zero $\hat{A}$ ignores the activation.

Complete$(\hat{A}, \hat{B}, X, Y)$*:* $\hat{A}$ *checks that* $Y \neq 0$ *and that she has an open session with identifier* $(\hat{A}, \hat{B}, X)$*. If any of these conditions fails* $\hat{A}$ *ignores the activation, otherwise she completes the session with session id* $(\hat{A}, \hat{B}, X, Y)$ *and the session key* $K = H(\pi(\hat{A}, \hat{B}, X, Y))$*.*
This represents the delivery of the second message in the protocol with the incoming value Y, (allegedly) the response from peer $\hat{B}$.

Remark. In normal circumstances the above activations are decided by the application invoking the protocol and by the incoming protocol messages. However, in our adversarial model we will analyze the protocol assuming that protocol activations and their inputs are chosen by the attacker. In particular, while the *normal* semantics of the *Respond* activation is that prior to this activation $\hat{B}$ was activated with an *Initiate* message with peer $\hat{A}$, and that $\hat{B}$ output Y in that activation, this does not actually have to be the case when the attacker $\mathcal{M}$ controls communications and the scheduling of activations. Indeed, $\mathcal{M}$ can generate a *Respond* activation for A regardless of the existence or not of an open session $(\hat{B}, \hat{A}, Y)$ at $\hat{B}$. The same holds for the *Complete* activation where the value Y could have been generated by $\hat{B}$ for this or other session, or simply never generated by $\hat{B}$.

Session state

For the time being we specify that the only values stored in a session state are the peer's identity $\hat{B}$, the outgoing and incoming DH values X, Y, and the session key K.

In particular, neither the exponent (dlog) x of the outgoing DH value X nor $\pi(\hat{A}, \hat{B}, X, Y)$ are stored in the session state. In other words, the latter values are not accessible to the attacker in a state-reveal query. Note that since our security model considers the long-term private key a to be stored separately (and, supposedly, more securely) than the ephemeral state information, then the computation of $\pi(\hat{A}, \hat{B}, X, Y)$ that uses the private key must be carried in the same module that stores this key (and therefore not part of the session state). Thus, only $K = H(\pi(\hat{A}, \hat{B}, X, Y))$ is to be output into the session state. On the other hand, forbidding the storage of the ephemeral exponent x in the session state is too restrictive (especially considering that such exponents may be chosen off-line to speed-up exponentiation) and therefore we will show in Section 7 (Lemma 29) that this restriction is not needed (the reason we separate the analysis of this property is that it relies on stronger assumptions).

Session identifiers and matching sessions

The session identifiers (in the form of a quadruple $(\hat{A}, \hat{B}, X, Y)$ for completed sessions) are significant in that they bind together incoming and outgoing messages and the identities of peers. In particular, a session with identifier $(\hat{A}, \hat{B}, X, Y)$ means that the session is held by $\hat{A}$, that X was the (unique) outgoing value generated by $\hat{A}$ for this session, and that the session was activated at $\hat{A}$ with peer $\hat{B}$ and incoming value Y.

With the above notation we obtain that if $(\hat{A}, \hat{B}, X, Y)$ is a complete session at $\hat{A}$ then its matching session (if it exists) is unique, its owner is $\hat{B}$ and its session-id is $(\hat{B}, \hat{A}, Y, X)$.

5.2 Proof of Security for the HMQV Protocol

We are now ready to prove the basic security of the HMQV protocol. See Section 6 for an analysis of additional security properties of the protocol.

Theorem 8 *Under the CDH assumption, the HMQV protocol* Π*, with its hash function modeled as a random oracle, is a secure key-exchange protocol (without PFS) in the Canetti-Krawczyk model.*

The PFS property (which holds in a weak form in HMQV) is further discussed and analyzed in Section 6 and Section 8.

The proof of the above theorem follows from the definition of secure key-exchange protocols outlined in Section 2 and the following two lemmas.

Lemma 9 *If two parties $\hat{A}, \hat{B}$ complete matching sessions, then their session keys are the same.*

Lemma 10 *Under the CDH assumption, there is no feasible adversary that succeeds in distinguishing the session key of an unexposed session with non-negligible probability.*

Lemma 9 follows immediately from the definition of matching sessions. That is, if $\hat{A}$ completes session (A, B, X, Y) and $\hat{B}$ completes the matching session $(\hat{B}, \hat{A}, Y, X)$ then $\hat{A}$ computes its session key as $H(\pi(\hat{A}, \hat{B}, X, Y))$ while $\hat{B}$ computes the same value as $H(\pi(\hat{B}, \hat{A}, Y, X))$.

The proof of Lemma 10 is far more involved and occupies the rest of this section. It consists of showing that given a successful KE-attacker $\mathcal{M}$ against protocol Π (i.e., $\mathcal{M}$ runs in polynomial-time and succeeds with non-negligible probability in the distinguishing game of Definition 1) one can build a successful forger against the dual signature DCR. Combining this with Theorem 7 we get the existence of an efficient CDH solver in contradiction to the CDH assumption.

In this section, we develop the proof of HMQV for the case that the attacker chooses test sessions in which the peers are different entities (in particular, with different public keys). Later in Section 6.3 we complete the proof by showing the security of the protocol when both peers are the same (in particular, this proves the resistance of HMQV to reflection attacks).

Terminology. By the session signature of a session $(\hat{A}, \hat{B}, X, Y)$ in Π we refer to the signature $\pi(\hat{A}, \hat{B}, X, Y)$. When $(\hat{A}, \hat{B}, X, Y)$ is the test session we refer to its session signature as the test signature. We usually denote the identifier of the test session as the quadruple $(\hat{A}, \hat{B}, X_0, Y_0)$.

5.2.1 Specialized attack goals

We start by observing that since the session key corresponding to the test session $(\hat{A}, \hat{B}, X_0, Y_0)$ is computed as $H(v)$, where v is the value of the test signature $\pi(\hat{A}, \hat{B}, X_0, Y_0)$, then the KE-attacker $\mathcal{M}$ has only two possible strategies to distinguish $H(v)$ from random (remember that we are modeling H as a random function):

Forging attack. At some point in its run $\mathcal{M}$ queries the function H on the value $\pi(\hat{A}, \hat{B}, X_0, Y_0)$. Clearly, this is possible only if $\mathcal{M}$ succeeds in computing (or learning) the test signature $\pi(\hat{A}, \hat{B}, X_0, Y_0)$.

Key-replication attack. $\mathcal{M}$ succeeds in forcing the establishment of a session (other than the test session or its matching session) that has the same key as the test session. In this case $\mathcal{M}$ can learn the test-session key by simply querying the session with the same key, and without having to learn the value of the test signature.

Therefore, in the sequel, instead of considering the regular distinguishing goal of a KE-attacker we will consider an attacker whose goal is to succeed (with non-negligible probability) in one of the above two attacks. We will show that if such an efficient attacker exists against Π then there also exists an efficient forger against the DCR scheme in contradiction (via Theorem 7) to the assumed infeasibility of CDH.

For simplicity of analysis we will consider the above two forms of attacks separately. We start by considering an attacker that wins the forging attack (Section 5.2.2) and then move to analyze key-replication attacks (Section 5.2.3).

5.2.2 Infeasibility of Forging Attacks

Let $\mathcal{M}$ be a forging attacker against Π. Namely, $\mathcal{M}$ interacts with protocol Π as a KE-attacker but does not run a distinguishing test. Instead, $\mathcal{M}$ ends its run either with a "fail" output or with a pair (sid, v) where sid is a session identifier and v a guess for the session signature of session sid. *We call $\mathcal{M}$ successful if it outputs, with non-negligible probability, a* correct *guess for the session signature of an* unexposed *session. The session for which $\mathcal{M}$ outputs a guess will be called the* test session*, and we refer to the corresponding session signature as the* test signature.

Consider a successful run of $\mathcal{M}$, and let $(\hat{A}, \hat{B}, X_0, Y_0)$ denote the test session for which $\mathcal{M}$ outputs a correct guess for the test signature $\pi(\hat{A}, \hat{B}, X_0, Y_0)$. By the convention on session identifiers, we know that the test session $(\hat{A}, \hat{B}, X_0, Y_0)$ is held by $\hat{A}$ (either as initiator or responder), $\hat{A}$ was activated in this session with peer identity $\hat{B}$, X_0 was the value output by $\hat{A}$ in the session, and Y_0 was the incoming value to this session. The generation of the value Y_0, delivered to $\hat{A}$ by $\mathcal{M}$, can fall under one of the following four cases:

C1. Y_0 was never output by $\hat{B}$ as its outgoing value in any of the sessions activated at $\hat{B}$, or $\hat{B}$ did output Y_0 as its outgoing value for some session s but it never computed the session key of s (e.g., $\hat{B}$ was activated as the initiator of s but was never activated with the *Complete* message to the session).

C2. Y_0 was generated by $\hat{B}$ in a session matching the test session, i.e., in session $(\hat{B}, \hat{A}, Y_0, X_0)$.

C3. Y_0 was generated at $\hat{B}$ during a session $(\hat{B}, \hat{A}^*, Y_0, X^*)$ with $\hat{A}^* \neq \hat{A}$ (and arbitrary X^*).

C4. Y_0 was generated at $\hat{B}$ during session $(\hat{B}, \hat{A}^*, Y_0, X^*)$ with $\hat{A}^* = \hat{A}$ but $X^* \neq X_0$.

Since we assume that $\mathcal{M}$ succeeds in its forgery attack with non-negligible probability then there is at least one of the above cases that happens with non-negligible probability in the successful runs of $\mathcal{M}$. Now, for each of the cases C1-C4 we build a forger $\mathcal{F}$ against the DCR scheme with the following property: if $\mathcal{M}$ succeeds with non-negligible probability in case Ci then the forger $\mathcal{F}$ built for case Ci succeeds with non-negligible probability in forging DCR signatures.

Important. The forgers that we build below interact with the signer $\hat{B}$ in the modified order of interaction as specified in Remark 4.3.

5.2.2.1 Forger $\mathcal{F}$ for Cases C1-C3

We start by presenting a first forger $\mathcal{F}$ that works in cases C1, C2 and C3. Namely, given an attacker $\mathcal{M}$ that guesses the test signature with non-negligible probability in one of these three cases, $\mathcal{F}$ will use $\mathcal{M}$ (as a subroutine) in order to forge DCR signatures also with non-negligible probability. In the description of $\mathcal{F}$ we will assume that whenever a session (other than the test session or its matching session) is exposed, the attacker $\mathcal{M}$ *receives the session signature value and not just the session key.* This simplifies our presentation but also shows a stronger result, namely, that in cases C1-C3 this extra information provided to $\mathcal{M}$ does not help in wining a forgery attack. The forger $\mathcal{F}$ that we construct is modeled after Definition 3 and Figure 3 with the modifications stated in Remark 4.3.

$\mathcal{F}$ actions are as follows:

1. The input to $\mathcal{F}$ consists of values $X_0, B \in_R G$ and an oracle $\hat{B}$ for DCR signatures under public key B. (That is, $\hat{B}$ is the signer, B its public key, and X_0 is the challenge under which $\mathcal{F}$ is trying to forge $\hat{B}$'s signature.) F's output consists of a (not necessarily correct) forgery against $\hat{B}$'s DCR signatures, or a "fail" symbol. In addition, $\mathcal{F}$ may abort its run without output (technically, this is equivalent to outputting "fail" but we differentiate between these two cases for the purpose of the analysis of $\mathcal{F}$).

2. $\mathcal{F}$ builds a virtual setting for the run of protocol Π under attacker $\mathcal{M}$, with n parties $P_1, P_2, \ldots, P_n$ in which each party is activated at most m times (n and m are polynomials in the security parameter).

3. $\mathcal{F}$ chooses $i, j \in_R \{1, \ldots, n\}, i \neq j,$[7] and $t \in_R \{1, \ldots, m\}$, and sets the public key of party P_j to be B. With these choices $\mathcal{F}$ is trying to guess which session will be chosen by $\mathcal{M}$ as the test session, and who the peers to the session will be. $\mathcal{F}$'s guess will be correct if $\mathcal{M}$ ends choosing as the test session the t-th session activated at party P_i (either as initiator or responder) and with P_j as the peer to that session. Since P_j is given the public key B, we will refer to P_j as $\hat{B}$ and to P_i as $\hat{A}$. Also, we will refer to the t-th session at P_i as the g-session ('g' for guess, namely, the session guessed by $\mathcal{F}$ as the test session to be chosen by $\mathcal{M}$).

4. $\mathcal{F}$ chooses private and public keys for each of the n parties in the protocol, except for P_j for whom the public key was set to B (and whose private key b is unknown to $\mathcal{F}$). We note that while $\mathcal{F}$ sets all these keys initially, the attacker is allowed to later choose new, arbitrary, public keys for corrupted parties.

5. $\mathcal{F}$ proceeds to run the protocol Π under the control of $\mathcal{M}$ who schedules all session activations as well as session exposures and party corruptions. With the exception of $\hat{B}$ (whose behavior we explain below) all actions of uncorrupted parties are performed by $\mathcal{F}$ who has full information about their private keys and internal states. Any message produced by an uncorrupted party is handed by $\mathcal{F}$ to $\mathcal{M}$.

6. When a session is activated at party $\hat{B}$ (whose private key b $\mathcal{F}$ does not know) then $\mathcal{F}$ uses its signing oracle (which we will also refer to as $\hat{B}$) to determine the actions of party $\hat{B}$ in the protocol. Specifically, when $\mathcal{M}$ activates a session at $\hat{B}$, either as responder or initiator, with peer identity $\hat{P}$ and incoming value X, then $\mathcal{F}$ feeds the signer $\hat{B}$ with $\hat{P}$ as the message to be signed and P as the public key (of $\hat{P}$) with respect to which the dual signature of $\hat{B}$ is to be generated. In response, $\mathcal{F}$ gets a value Y from the signer $\hat{B}$. $\mathcal{F}$ then hands $\mathcal{M}$ the value Y as the outgoing value from party $\hat{B}$. (Note that $\mathcal{F}$ does not request the full signature from $\hat{B}$ at this point; this will be done only in case that a state-reveal or session-key query is later performed by $\mathcal{M}$ on the session).

7. When $\mathcal{M}$ issues a state-reveal or session-key query to a session, $\mathcal{F}$ responds to it with the value of the corresponding session signature (if already determined). If the queried session is one held by $\hat{B}$ then $\mathcal{F}$ queries the session signature from $\hat{B}$ (by presenting to the signing oracle the triple $(Y, \text{peer-id}, X)$ corresponding to the queried session – see Remark 4.3), and returns the value of the signature to $\mathcal{M}$.

[7] We deal with the case $i = j$, i.e., reflection attacks, in Section 6.3.

8. Upon corruption of a party (other than $\hat{B}$) by $\mathcal{M}$, $\mathcal{F}$ provides $\mathcal{M}$ with the private key of the party and any state information for current sessions and unexpired session keys at the party (specifically, $\mathcal{F}$ provides $\mathcal{M}$ with the session signatures corresponding to these sessions, not just the session keys). From the moment of corruption all actions of a corrupted party are decided by $\mathcal{M}$ (including the possible "registration" of a new public key for the party).

9. When $\mathcal{M}$ activates the t-th session at $\hat{A}$ (i.e., the g-session), if the peer in the activation is not $\hat{B}$ then $\mathcal{F}$ aborts. Else, $\mathcal{F}$ provides $\mathcal{M}$ with the value X_0 (from $\mathcal{F}$'s input) as the outgoing value of $\hat{A}$ in that session (this is done regardless of whether $\hat{A}$ was activated as initiator or responder).

10. In addition in any of the following cases $\mathcal{F}$ aborts its run:

    (a) $\mathcal{M}$ halts with a test session different than the g-session.

    (b) $\mathcal{M}$ corrupts either $\hat{A}$ or $\hat{B}$. (Later, we will see that we can actually let $\mathcal{M}$ learn $\hat{A}$'s private key, but for the moment it is simpler to assume that $\hat{A}$ is never corrupted.)

    (c) $\mathcal{M}$ exposes the g-session via a state-reveal or session-key query.

    (d) $\mathcal{M}$ exposes the session matching to the g-session (if such session exists) via a state-reveal or session-key query.

11. If $\mathcal{M}$ halts with the g-session as its test session (in this case the test session has identifier $(\hat{A}, \hat{B}, X_0, Y_0)$ for some value Y_0) and with a guess $\tilde{\pi}_0$ for the test signature, then $\mathcal{F}$ outputs the triple $(\hat{A}, Y_0, \tilde{\pi}_0)$ as a forgery of $\hat{B}$'s DCR signature on message $m = \hat{A}$ (and challenge X_0). Else, if $\mathcal{M}$ outputs "fail" (i.e., no test session chosen) then $\mathcal{F}$ outputs "fail" as well.

Lemma 11 *Assume that $\mathcal{M}$ has a non-negligible probability to correctly guess the test signature when either of cases C1-C3 holds, then the above forger $\mathcal{F}$ succeeds in forging $\hat{B}$'s* DCR *signatures with non-negligible probability.*

The lemma is proved through Claims 12 and 13.

Claim 12 *Let $\mathcal{M}$ be a forging attacker against protocol* Π *which succeeds with non-negligible probability when at least one of cases C1-C3 holds. Then there is a non-negligible probability that a run of $\mathcal{M}$ under $\mathcal{F}$ succeeds (i.e., ends with a correct guess for the value of the unexposed test signature).*

Proof: By fixing a set of random coins for the attacker $\mathcal{M}$ and for each of the parties in protocol Π, together with a fixing of the random functions $\bar{H}$ and H, we obtain a deterministic execution of protocol Π. In particular, these fixed values fully determine the actions and view of $\mathcal{M}$ in that execution. In the sequel we refer to such a fixed execution as a run of $\mathcal{M}$. In the case that $\mathcal{M}$ is run by $\mathcal{F}$ (as specified in the above description of $\mathcal{F}$), then it is $\mathcal{F}$ that provides all of the random coins to $\mathcal{M}$ and to the protocol parties except for party $\hat{B}$. The latter has its own random coins as represented by the random coins used by the signing oracle $\hat{B}$. These sets of random coins determine a run of $\mathcal{M}$ under $\mathcal{F}$ that may be different than the run of $\mathcal{M}$ with the same coins in a real execution of the protocol.

We say that a run of $\mathcal{M}$ under $\mathcal{F}$ is *perfect* if it is identical to the run of the real $\mathcal{M}$ under the same coins set by $\mathcal{F}$ and by oracle $\hat{B}$. We claim that any run of $\mathcal{M}$ under $\mathcal{F}$ in which $\mathcal{F}$ does not abort is perfect. Indeed, one can see that the simulation of the run of $\mathcal{M}$ by $\mathcal{F}$ is perfect up to the point that $\mathcal{F}$ aborts (and thus a run in which $\mathcal{F}$ does not abort is perfect). To see this, consider first the simulation by $\mathcal{F}$ of actions related to uncorrupted players other than $\hat{B}$. For these players $\mathcal{F}$ has full information and thus it runs all their actions (session activations, session exposures and party corruptions) exactly as in a real interaction with $\mathcal{M}$. One possible deviation happens when $\mathcal{M}$ activates the g-session at $\hat{A}$: in this case rather than choosing an outgoing value $X = g^x$ for the session, $\mathcal{F}$ uses X_0, received in $\mathcal{F}$'s input, as the outgoing value. However since $X_0 \in_{\mathrm{R}} G$ then it has the identical distribution of a real X; moreover, since as long as $\mathcal{F}$ does not abort the g-session is not queried, then $\mathcal{F}$ does not need to take any action that would require the knowledge of the logarithm of X_0.

It remains to consider the simulation by $\mathcal{F}$ of party $\hat{B}$ for whom $\mathcal{F}$ does not know the private key b, and therefore only runs it by using the signing oracle as a black box. Note, first that $\mathcal{F}$ does not need to provide b to the attacker since a corruption of $\hat{B}$ triggers an abort by $\mathcal{F}$. Therefore, $\mathcal{F}$ only needs to simulate $\hat{B}$'s actions related to session activations and session exposures. However these two forms of actions are perfectly simulated by $\mathcal{F}$ using the signing oracle $\hat{B}$. The resultant actions correspond exactly to the behavior of a real player $\hat{B}$ with public key B and random coins as those used by the oracle $\hat{B}$.

So we've seen that non-aborting runs of $\mathcal{F}$ result in perfect runs of $\mathcal{M}$. On the other hand, non-aborting runs of $\mathcal{F}$ happen only when $\mathcal{M}$ chooses its test session equal to the g-session (or when $\mathcal{M}$ fails without choosing a test session). Let's consider the triples (i, j, t) as used by $\mathcal{F}$ (in step 3) to choose the g-session. Let γ denote one specific triple (such a triple defines a unique session which may or may not be activated in a given run of $\mathcal{M}$). From the above argument we have that all runs of $\mathcal{M}$ that end with the choice of session γ as the test session of $\mathcal{M}$ will result in perfect runs under $\mathcal{F}$ *provided that* $\mathcal{F}$ *chooses* γ *as its g-session* (since in this case $\mathcal{F}$ will not abort). In particular, this implies that for every value of γ:

$$\begin{aligned} &\mathit{Prob}\,(\mathcal{M} \text{ succeeds in session } \gamma \text{ under the run of } \mathcal{F} : \mathcal{F} \text{ chooses } \gamma \text{ as its g-session}) \\ &\quad = \mathit{Prob}\,(\mathcal{M} \text{ succeeds in session } \gamma). \end{aligned}$$

By "$\mathcal{M}$ succeeds in session γ" we mean that $\mathcal{M}$ outputs a correct guess for the session signature corresponding to session γ.

Since a run of $\mathcal{M}$ is successful (i.e., ends with a correct guess for the test signature) if and only if there is an (unexposed) session γ activated at this run for which $\mathcal{M}$ outputs a correct forgery (and since there are at most n^2m possible values of γ) we get:

$$\begin{aligned} &\mathit{Prob}\,(\text{Successful run of } \mathcal{M} \text{ under } \mathcal{F}) = \\ &\quad = \sum_{\gamma} \mathit{Prob}\,(\mathcal{F} \text{ chooses } \gamma \text{ as the g-session } \mathit{AND}\ \mathcal{M} \text{ succeeds in } \gamma \text{ under } \mathcal{F}) \\ &\quad = \sum_{\gamma} \mathit{Prob}\,(\mathcal{M} \text{ succeeds in } \gamma \text{ under } \mathcal{F} : \mathcal{F} \text{ chooses } \gamma \text{ as the g-session}) \ * \\ &\qquad\quad * \mathit{Prob}\,(\ F \text{ chooses } \gamma \text{ as the g-session}) \\ &\quad \geq (n^2m)^{-1} \sum_{\gamma} \mathit{Prob}\,(\mathcal{M} \text{ succeeds in } \gamma) \\ &\quad = (n^2m)^{-1} \mathit{Prob}\,(\mathcal{M} \text{ succeeds}) \end{aligned}$$

Since by assumption, $Prob\,(\mathcal{M}$ succeeds) is non-negligible, when at least one of the cases C1-C3 holds, then the above bound on $Prob\,$(Successful run of $\mathcal{M}$ under $\mathcal{F}$) is non-negligible.

□

Claim 13 *If in a run of $\mathcal{F}$, $\mathcal{M}$ outputs a correct guess for the test signature, then the output of $\mathcal{F}$ in that run is a correct and valid forgery against $\hat{B}$'s* DCR *signatures.*

Proof: By construction, $\mathcal{F}$ outputs an attempted forgery against $\hat{B}$'s signatures each time that $\mathcal{M}$, as run by $\mathcal{F}$, completes its run and outputs a guess $\tilde{\pi}_0$ for the test signature $\pi_0 = \pi(\hat{A}, \hat{B}, X_0, Y_0)$. More precisely, in this case $\mathcal{F}$ outputs a forgery triple $\tau = (m = \hat{A}, Y_0, \tilde{\pi}_0)$ under the input challenge value X_0. Clearly, the forgery is correct if $\tilde{\pi}_0 = \pi_0$ which is the case by the claim's hypothesis. Therefore, what remains to be shown is that the triple τ is a valid forgery, namely, that $Y_0 \neq 0$ and the pair $(m = \hat{A}, Y = Y_0)$ never appeared in a signature issued by $\hat{B}$ in any of its invocations by $\mathcal{F}$. The first condition, $Y_0 \neq 0$, holds since otherwise this value would have been rejected by $\hat{A}$ as the incoming value to the g-session. As for the second condition, let's assume that $\tilde{\pi}_0 = \pi_0$ and examine its validity under each of the three cases C1-C3.

Case C1: In this case the value Y_0 was never output in any of the signatures issued by $\hat{B}$ and therefore τ is clearly a valid forgery.

Case C2: Here, Y_0 was generated by $\hat{B}$ in session $(\hat{B}, \hat{A}, Y_0, X_0)$. However this session, being matching to the test session, was never queried by $\mathcal{M}$ (or $\mathcal{F}$ would have aborted without outputting a forgery), which in turn implies that $\mathcal{F}$ never queried $\hat{B}$ for the value of $\pi(\hat{B}, \hat{A}, Y_0, X_0)$. Note that, by construction, $\mathcal{F}$ *never* queries the signature corresponding to a session in $\hat{B}$ *except if this session is queried, via state reveal or session-key queries, by* $\mathcal{M}$. Since Y_0 was generated by $\hat{B}$ only for the above session (except for a negligible probability of accidental repetition) then no signature issued by $\hat{B}$ has this value of Y_0. Thus, τ is a valid forgery.

Case C3: The value Y_0 was generated by $\hat{B}$ in relation to a session $(\hat{B}, \hat{A}^*, Y_0, X^*)$ with $\hat{A}^* \neq \hat{A}$. Thus, if $\mathcal{M}$ queried this session then $\hat{B}$ did produce the signature $\pi(\hat{B}, \hat{A}^*, Y_0, X^*)$. However this is a signature of $\hat{B}$ with Y_0 on message $m = \hat{A}^*$ which is different than the message $m = \hat{A}$ in the forgery τ. Hence τ is valid in this case too

Summarizing, for each of the above cases, if $\mathcal{M}$ succeeds with non-negligible probability in that case, then with non-negligible probability $\mathcal{F}$ outputs a forgery triple τ which is correct and valid. □

5.2.2.2 Forger $\mathcal{F}'$ for Case C4

We now define a forger $\mathcal{F}'$ that will succeed in forging $\hat{B}$'s DCR signatures provided that there is a non-negligible probability that $\mathcal{M}$ succeeds in a forgery attack under case C4. Recall that this case happens when $\mathcal{M}$ outputs a correct guess π_0 for the signature of the test session $(\hat{A}, \hat{B}, X_0, Y_0)$ and the value Y_0 was generated by $\hat{B}$ in another session $(\hat{B}, \hat{A}, Y_0, X^*)$ with $X^* \neq X_0$. We build the forger $\mathcal{F}'$ following the same lines of the design of $\mathcal{F}$, but with some important differences that we specify next.

1. In addition to choosing random values i, j, t as $\mathcal{F}$ does, $\mathcal{F}'$ also chooses a number $l \in_{\mathrm{R}} \{1, \ldots, m\}$. As before we denote P_i and P_j by $\hat{A}$ and $\hat{B}$, respectively.
The choice of i, j, t represents a guess by F' that the test session will be chosen by $\mathcal{M}$ as

the t-th session activated at P_i and that P_j will be the peer to this session; the choice of l represents a further guess that the l-th session activated at $\hat{B}$ will be the session in which $\hat{B}$ generated Y_0, i.e. this session will have identifier $(\hat{B}, \hat{A}, Y_0, X^*)$ for some $X^* \neq X_0$.

2. $\mathcal{F}'$ aborts in the cases that $\mathcal{F}$ does, plus in the following two cases: (i) the l-th activation at $\hat{B}$ has a peer other than $\hat{A}$ or it has $X^* = X_0$, and (ii) the incoming value fed by $\mathcal{M}$ to $\hat{A}$ in its t-th session is different than the value Y_0 output by $\hat{B}$ in its l-th activation. (This guarantees that if $\mathcal{F}'$ does not abort its run then the l-th activation of $\hat{B}$ resulted in a session $(\hat{B}, \hat{A}, Y_0, X^*)$ for some value $X^* \neq X_0$.)

3. The rest of $\mathcal{F}'$ is the same as $\mathcal{F}$ except that in the l-th activation at $\hat{B}$, $\mathcal{F}'$ does not query the signer $\hat{B}$ but instead it chooses a random Y_0 and provides it to $\mathcal{M}$ as $\hat{B}$'s outgoing value for this session. If at any point $\mathcal{M}$ queries the session $(\hat{B}, \hat{A}, Y_0, X^*)$ (via a state reveal or session-key query) then $\mathcal{F}'$ chooses a random value in $\{0,1\}^k$ (the range of H) and answers $\mathcal{M}$'s query with that value. (Note that $\mathcal{F}'$ never queries the signature $\pi(\hat{B}, \hat{A}, Y_0, X^*)$.)

4. If $\mathcal{M}$ halts without having queried session $(\hat{B}, \hat{A}, Y_0, X^*)$ then $\mathcal{F}'$ halts with same result as $\mathcal{M}$, namely, if $\mathcal{M}$ fails so does F', and if $\mathcal{M}$ outputs a guess $\tilde{\pi}_0$ for $\pi(\hat{A}, \hat{B}, X_0, Y_0)$ then $\mathcal{F}'$ outputs the triple $(\hat{A}, Y_0, \tilde{\pi}_0)$ as its forgery of $\hat{B}$'s signature on message $m = \hat{A}$ (under challenge X_0).

5. If $\mathcal{M}$ halts after having queried session $(\hat{B}, \hat{A}, Y_0, X^*)$, F' chooses a random bit β and proceeds as follows:

   (a) Case $\beta = 0$. $\mathcal{F}'$ halts and its output is determined exactly as in Step 4 above.

   (b) Case $\beta = 1$. **Rewinding step:** $\mathcal{F}'$ rewinds $\mathcal{M}$ to its state at the time it queried session[8] $(\hat{B}, \hat{A}, Y_0, X^*)$. This time, however, instead of answering this query by a random value, $\mathcal{F}'$ chooses at random one of the queries to H produced by $\mathcal{M}$, say v, and responds to the $(\hat{B}, \hat{A}, Y_0, X^*)$ query with $H(v)$. (The query v is chosen among all queries by $\mathcal{M}$ to H including queries made after $\mathcal{M}$ queried the session $(\hat{B}, \hat{A}, Y_0, X^*)$ in the original run.) From this point $\mathcal{F}'$ continues the normal running of $\mathcal{M}$, which may end with $\mathcal{F}'$ aborting or with $\mathcal{M}$ halting. In the latter case, the output of $\mathcal{F}'$ is determined as in Step 4 above.

   Note: In the above run of $\mathcal{M}$ by $\mathcal{F}'$ it may happen that if $\mathcal{M}$ queries the session $(\hat{B}, \hat{A}, Y_0, X^*)$ the response by $\mathcal{F}'$ is different than the actual value $H(\pi(\hat{B}, \hat{A}, Y_0, X^*))$. This discrepancy relative to the real runs of $\mathcal{M}$ may cause $\mathcal{M}$ to run beyond the normal time bounds of $\mathcal{M}$. Therefore, to be precise, we can specify that if $\mathcal{M}$ exceeds a certain (polynomial) running time then $\mathcal{F}'$ stops and outputs failure.

Lemma 14 *Assume that $\mathcal{M}$ has a non-negligible probability to correctly guess the test signature when case C4 holds, then forger $\mathcal{F}'$ succeeds in forging $\hat{B}$'s* DCR *signatures with non-negligible probability.*

The lemma is proved through Claims 15 and 16.

[8] Note that by construction of $\mathcal{F}'$ the rewinding step is performed only in cases where $\mathcal{M}$ actually queried (B, A, Y_0, X^*) during its run.

Claim 15 *Let $\mathcal{M}$ be a forging attacker against protocol* Π *which succeeds with non-negligible probability in case C4, then there is a non-negligible probability that a run of* $\mathcal{M}$ *under* $\mathcal{F}'$ *succeeds (i.e., ends with a correct guess for the value of the unexposed test signature).*

Proof: As in Claim 12 we refer to a "run of $\mathcal{M}$" as a fixed (deterministic) execution of the attacker $\mathcal{M}$ when we fix a set of random coins for $\mathcal{M}$, for all the parties in the protocol, and for the random functions $\bar{H}$ and H.

We will say that a run of $\mathcal{M}$ is of `type-C4` if $\mathcal{M}$ outputs a guess (not necessarily a correct one) for the session signature of an unexposed test session $(\hat{A}, \hat{B}, X_0, Y_0)$ (chosen by $\mathcal{M}$) in which Y_0 was generated by $\hat{B}$ during a session $(\hat{B}, \hat{A}, Y_0, X^*)$ with $X^* \neq X_0$. Note that by the Claim's hypothesis, the attacker $\mathcal{M}$ has a non-negligible probability to succeed (i.e., guess correctly) in a run of type-C4. In particular, this implies that type-C4 runs occur with non-negligible probability in the runs of $\mathcal{M}$. Also note that in type-C4 runs the session $(\hat{B}, \hat{A}, Y_0, X^*)$ is well-defined and unique (up to a negligible probability that $\hat{B}$ chose the same Y_0 for two different sessions).

We denote the test session $(\hat{A}, \hat{B}, X_0, Y_0)$ by s_0, and the session $(\hat{B}, \hat{A}, Y_0, X^*)$ by s^*. The test signature $\pi(\hat{A}, \hat{B}, X_0, Y_0)$ is denoted by π_0 and the session signature $\pi(\hat{B}, \hat{A}, Y_0, X^*)$ of session s^* by π^*. We will say that $\mathcal{M}$ queries session s^* if $\mathcal{M}$ issues a state-reveal or session-key query against s^*. We say that $\mathcal{M}$ queries H on π^* if $\mathcal{M}$ presents the value π^* explicitly as an input to H (therefore querying s^* is not the same as querying H on π^* even though the answer to both queries is the same).

We divide the type-C4 runs of $\mathcal{M}$ into three classes:

1. $\mathcal{M}$ does not query the session s^*.

2. $\mathcal{M}$ queries session s^* but does query H on π^*.

3. $\mathcal{M}$ queries s^* and also queries H on π^*.

We proceed to analyze what happens to these runs in the executions of $\mathcal{M}$ under $\mathcal{F}'$. *For the time being we are going to consider only executions of* $\mathcal{F}'$ *in which* $\mathcal{F}'$ *does not abort* (i.e., in which $\mathcal{M}$'s run is of type-C4 and $\mathcal{F}'$ guessed correctly the s_0 and s^* sessions). First we need two observations from which we can see that the notion of a run of C4-type is well defined for runs of $\mathcal{M}$ under $\mathcal{F}'$.

The first is that following a similar argument as in the proof of Claim 12, in the non-aborting runs of $\mathcal{F}'$ the simulation of the protocol actions carried by $\mathcal{F}'$ for $\mathcal{M}$ are perfect except for (i) the response to the query to session s^*, if such a query is presented by $\mathcal{M}$, and (ii) the subsequent rewinding step, if performed. For the second observation we consider the runs of $\mathcal{M}$ under $\mathcal{F}'$ that precede the rewinding step (or those in which this step is not performed at all). Note that in runs in which $\mathcal{M}$ queries s^* before it queries H on π^* (this includes runs where $\mathcal{M}$ does not query H on π^* at all), the probability that $\mathcal{M}$ later queries H on π^* is the same whether the response to the s^* query was the real $H(\pi^*)$ or a random value (indeed, at this point of the run these two responses are indistinguishable given the current view of $\mathcal{M}$). Similarly, if $\mathcal{M}$ first queries H on π^*, then the probability that it will later query s^* is the same regardless of the random value used to answer $H(\pi^*)$.

From the above two observations we obtain that the probability that $\mathcal{M}$ queries the session s^* and/or queries H on π^* is the same in the real runs of $\mathcal{M}$ than in the runs under $\mathcal{F}'$ (and before carrying the rewinding step). Thus, the probability that a run of $\mathcal{M}$ under $\mathcal{F}'$ (before the rewinding step) falls into any one of the three classes of C4-type runs is the same as the probability that a real $\mathcal{M}$ run fall in this case.

Let's now consider how the behavior of $\mathcal{M}$ when run under $\mathcal{F}'$ differs from a real run of $\mathcal{M}$ in each of the three classes of C4-type runs. (Remember that we are considering only non-aborting executions of $\mathcal{F}'$.) In the first class the run of $\mathcal{M}$ by $\mathcal{F}'$ is identical to a real run (since none of the differences (i) or (ii) exist). In the second case, if $\beta = 0$ then the run of $\mathcal{M}$ under $\mathcal{F}'$ is identical to a real run since as long as $\mathcal{M}$ does not query H on π^* then the random response of $\mathcal{F}'$ to the s^* query is indistinguishable from any other random value. Finally, for the third class we consider the case that $\beta = 1$ and $\mathcal{F}'$ happens to choose the query v by $\mathcal{M}$ such that $v = \pi^*$ (there is a non-negligible probability, at least $1/Q$, that this happens since we are considering the case that at least one of the Q queries made by $\mathcal{M}$ to H during its run was the value π^*). In this case, if we consider the "hybrid" run by $\mathcal{M}$ as the one composed of the execution preceding the query by $\mathcal{M}$ of session s^* followed by the execution of $\mathcal{M}$ after the rewinding step, we get that this run is identical to the corresponding run of the real $\mathcal{M}$ (in which the query to s^* is answered with the correct value $H(\pi^*)$).

In all, we have that for each class of type-C4 runs the probability of such runs under $\mathcal{F}'$ is the same as in a real execution of $\mathcal{M}$, and that there is a non-negligible probability (over the choice of β and of the query v in the rewinding step) that the behavior of $\mathcal{M}$ in these runs under $\mathcal{F}'$ is identical to the corresponding runs in a real execution of $\mathcal{M}$.

Thus, since there is a non-negligible probability that a type-C4 run of (the real) $\mathcal{M}$ will be successful, then there must be at least one class of C4-type runs in which the probability of success of $\mathcal{M}$ is also non-negligible. But then, by the above argument, the latter is true also for this class of runs under $\mathcal{F}'$. We thus conclude that, in the non-aborting runs of $\mathcal{F}'$, $\mathcal{M}$ as run by $\mathcal{F}'$ has non-negligible probability of success.

All we need in order to conclude the proof is to observe that, by an argument similar to the one in the proof of Claim 12, for each type-C4 run of $\mathcal{M}$ there is a non-negligible probability (i.e., 1 over the number of quadruples (i, j, t, l) chosen by $\mathcal{F}'$ in Step 1), that this run under $\mathcal{F}'$ is non-aborting. Thus, the conditioning on non-aborting runs in the above arguments holds with non-negligible probability, and then the overall probability that the run of $\mathcal{M}$ under $\mathcal{F}'$ outputs a successful forgery is non-negligible.

□

Claim 16 *If in a run of $\mathcal{F}'$, $\mathcal{M}$ outputs a correct guess for the test signature, then the output of $\mathcal{F}'$ in that run is a correct and valid forgery against $\hat{B}$'s* DCR *signatures.*

Proof: In the case that $\mathcal{M}$ outputs the correct guess π_0 for the test session $(\hat{A}, \hat{B}, X_,Y_0)$ under a run of $\mathcal{F}'$, $\mathcal{F}'$ outputs a forgery against $\hat{B}$'s signatures of the form $\tau = (m = \hat{A}, Y_0, \pi_0)$, under the input challenge value X_0, which is correct since $\mathcal{M}$'s guess was correct. Therefore, it remains to show that when case C4 holds this forgery is also valid. That is, that $Y_0 \neq 0$ and the pair $(m = \hat{A}, Y = Y_0)$ never appeared in a signature issued by $\hat{B}$ in any of its invocations by $\mathcal{F}'$. The first condition, $Y_0 \neq 0$, holds since otherwise this value would have been rejected by $\hat{A}$ as the incoming value to the g-session. As for the second condition, we know that the only session in which $\hat{B}$ used Y_0 is the session $s^* = (\hat{B}, \hat{A}, Y_0, X^*)$. Thus the only signature that could have possibly used Y_0 is the session signature $\pi^* = \pi(\hat{B}, \hat{A}, Y_0, X^*)$. However, this signature was never queried by $\mathcal{F}'$. Indeed, even if $\mathcal{M}$ queried the session, $\mathcal{F}'$ answered this query by a random value (Step 3) or by a previous output of H (Step 5b), with no invocation to $\hat{B}$. Thus the forgery triple τ is a correct and valid forgery by $\mathcal{F}'$ against $\hat{B}$'s signatures. □

5.2.3 Infeasibility of Key-Replication Attacks

As introduced and motivated in Section 5.2.1 a successful key-replication attack is one in which the attacker, acting as a KE attacker, has as its goal to force the establishment of two different, non-matching, sessions that output the same session key, and in which one of the two sessions (we will refer to it as the "test session") is unexposed. Note that if an attacker succeeds in such an attack against Π, it can easily find the session key of the test session by querying the second session containing the same key (this is allowed since the sessions are non-matching).[9].

In this section we prove that key-replication attacks are infeasible against protocol Π by showing that such a successful attacker would contradict the security of DCR signature. For this we use the same forgers $\mathcal{F}$ and $\mathcal{F}'$ built in Section 5.2.2. Specifically, we show the following:

Lemma 17 *If an efficient attacker, $\mathcal{M}$, succeeds in a key-replication attack against Π with non-negligible probability then one of forgers F or F' constructed above is a successful forger against the* DCR *signature scheme.*

Proof: Assume that attacker $\mathcal{M}$ is successful in a replication attack against the (unexposed) test session denoted by $s = (\hat{A}, \hat{B}, X_0, Y_0)$. Namely, $\mathcal{M}$ succeeds in establishing a session $s' = (\hat{A}', \hat{B}', X', Y')$ which has the same key as $(\hat{A}, \hat{B}, X_0, Y_0)$ (and this session is different than $(\hat{A}, \hat{B}, X_0, Y_0)$ and $(\hat{B}, \hat{A}, Y_0, X_0)$). This means that (except of a negligible probability of collision in the output of H) $\pi(\hat{A}', \hat{B}', X', Y') = \pi(\hat{A}, \hat{B}, X_0, Y_0)$.

Let's now consider the four cases C1-C4, described in Section 5.2.2, concerning the generation of the value Y_0 in the test session s. Since these four cases cover all possible ways to generate Y_0 during the protocol, then there must be (at least) one of these cases in which $\mathcal{M}$ wins the key-replication attack with non-negligible probability. Assume, first, that this non-negligible probability of success holds for any of the first three cases C1-C3, and consider the forger $\mathcal{F}$ built above and its interaction with the key-replication attacker $\mathcal{M}$. Recall that, by construction, $\mathcal{F}$ provides $\mathcal{M}$ with the session signatures of exposed sessions (not just the session keys). Therefore, if $\mathcal{M}$ is able to succeed in a key-replication attack then it can query the session s' (which $\mathcal{M}$ is allowed to expose) and obtain the signature $\pi' = \pi(\hat{A}', \hat{B}', X', Y')$ from $\mathcal{F}$. But this means that $\mathcal{M}$ is able to find the test signature $\pi(\hat{A}, \hat{B}, X_0, Y_0)$ (which is the same as π') without exposing the test session or its matching session. But as we showed, in this case $\mathcal{F}$ succeeds in outputting a valid forgery against $\hat{B}$'s signatures.

Let's now consider case C4. Observing the construction of $\mathcal{F}'$, we note that $\mathcal{F}'$ provides $\mathcal{M}$ with the session signatures (not just the session keys) for any exposed session other than session $(\hat{B}, \hat{A}, Y_0, X^*)$. Thus, by the same argument above, if $\mathcal{M}$ is able to force the same key as the test session in a non-matching session other than $(\hat{B}, \hat{A}, Y_0, X^*)$ then $\mathcal{M}$ is able to win the forging game, and $\mathcal{F}'$ succeeds as a forger against $\hat{B}$'s signatures. We are then left with the possibility that the session $(\hat{B}, \hat{A}, Y_0, X^*)$ had the same key as $(\hat{A}, \hat{B}, X_0, Y_0)$. This however is not possible since for $X^* \neq X_0$ it must be that $\pi(\hat{B}, \hat{A}, Y_0, X^*) \neq \pi(\hat{B}, \hat{A}, Y_0, X_0) = \pi(\hat{A}, \hat{B}, X_0, Y_0)$ and thus the two session keys are different (except for a negligible probability of collision in the output of H on these two different input values). □

This completes the proof of Lemma 10 which, together with Lemma 9 and Lemma 24 (the latter proven in the next section), completes the proof of Theorem 8.

[9] Formally, key-replication attacks can be defined by requiring that the attacker explicitly outputs the identifiers of the two non-matching sessions that have the same key, or equivalently, i.e. up to a polynomial factor in success probability, one may just require that these sessions exist at the end of a successful run of the attacker.

6 Further Security Properties of the HMQV Protocol

In the previous section we have proved that the HMQV protocol enjoys the essential security properties captured by the formal model of [11] (and outlined in Section 2). This definition guarantees the essential security properties of a key-exchange protocol, namely, that a party that completes a session has the guarantee that (i) if the peer to the session is uncorrupted then the session key is not known to anyone except this peer, and (ii) if the peer completes a matching session then the two have the same shared key. These properties are guaranteed even if the attacker has corrupted other parties in the system and/or the attacker have learned secret information from other sessions. Most importantly, as shown in [11], this security definition is all that is needed to ensure that the communications between the session peers, when protected with the exchanged key (via symmetric encryption and MAC function), are secure.

In some applications and scenarios, however, there are additional properties of a key-exchange protocol that one may be interested to have. In this section we consider the properties and examine if they apply or not to HMQV.

As a result we have the following comprehensive extension of Theorem 8.

Theorem 18 *Under the CDH assumption and in the random oracle model, the HMQV protocol as defined in Section 5.1 is secure in the Canetti-Krawczyk model (outlined in Section 2) and in addition it enjoys the following properties: Resistance to KCI attacks, weak forward secrecy, and resilience to the leakage of ephemeral secret DH exponents (the last property proven in Section 7 under the Gap Diffie-Hellman and KEA1 assumptions).*

The definition and proof of the above additional properties is presented in the following subsections. Resilience to leakage of ephemeral DH exponents requires a more extensive treatment and is presented in Section 7.

6.1 Resistance to KCI Attacks

A basic principle guiding any security formulation for key-exchange protocols is that the leakage of ephemeral information associated with a particular session (e.g., its session key or state) does not compromise other sessions. As pointed out before, this property is indeed ensured by our security definition. On the other hand, this definition provides no guarantee for sessions exchanged by, or with, corrupted parties. Yet, in practice, it is important to confound the damage of key leakage even when this leakage concerns the long-term private key of a party. In this, and next subsection, we discuss two security notions that extend our basic model and deal precisely with limiting the damage of leakage of long term private keys.

We start by observing that when an attacker learns the private key of a party $\hat{A}$, our model considers that party to be under the full control of the attacker, and all the sessions established by that party are deemed insecure. This, however, is a simplification of practical scenarios in which it is possible that the private key of $\hat{A}$ is compromised, yet the attacker does not actively control $\hat{A}$ (e.g., it does not have access to $\hat{A}$'s computer). In such a situation, while there is nothing to prevent the attacker from running its own key exchange session in the name of $\hat{A}$, one would like to ensure the security of the sessions established by $\hat{A}$ itself while not being actively controlled by the attacker.

An example of this type of security is the so called *resistance to key-compromise impersonation (KCI).* This notion has been postulated in many works as a desirable property of (public-key based) key-exchange protocols and, in particular, it has been stated as one of the advantages of the MQV protocol over other protocols. Informally (as this property has not been formalized before) a key-compromise impersonation is deemed successful if the attacker, knowing the private key of a party $\hat{A}$ is able to "impersonate" other, uncorrupted, parties to $\hat{A}$. Here, we first formalize this notion and then prove it to hold in the case of HMQV.[10] For this we first introduce the following definition:

Definition 19 *We say that a completed session of a key-exchange protocol is* clean *if the attacker did not have access to the session's state at the time of session establishment, nor it issued a session-key query against the session after completion.*

When considering sessions at uncorrupted parties the definition of clean session is equivalent to saying that the attacker did not issue a state-reveal query while the session was incomplete or a session-key query after completion. However, when considering an attacker that learned the long-term private key of a party, this notion is more general and it implies that the attacker was not actively controlling or impersonating the party during the session establishment (neither by making any choices on behalf of that party in that session or eavesdropping into the session's state). Note that in the case of HMQV, a clean session is one in which the attacker does not choose the outgoing value of the session, nor it issues a session-key query against it.

Definition 20 *We say that a KE-attacker $\mathcal{M}$ that has learned the private key of party $\hat{A}$ succeeds in a* Key-compromise impersonation (KCI) *attack against $\hat{A}$, if $\mathcal{M}$ is able to distinguish from random the session key of a complete session at $\hat{A}$ for which the session peer is uncorrupted and the session and its matching session (if it exists) are clean.*

In other words, the definition says that as long as the attacker is not actively controlling or observing the secret choices made for the generation of the session then, even the knowledge of $\hat{A}$'s private key does not allow $\mathcal{M}$ to compromise the session key. In particular, in such a protocol $\mathcal{M}$ cannot impersonate an uncorrupted party $\hat{B}$ to $\hat{A}$ in a way that allow $\mathcal{M}$ to learn (any information about) the resultant session key.

Lemma 21 *Under the CDH assumption, the HMQV protocol resists key-compromise impersonation (KCI) attacks.*

Proof: All is needed is to note that the proof of HMQV in Section 5.2 holds even if we assume that the attacker $\mathcal{M}$ is given the private key of $\hat{A}$. The only change to the proof is that in the description of forger $\mathcal{F}$ (step 10b) we now remove the corruption of $\hat{A}$ as a reason for $\mathcal{F}$ to abort. The proof remains valid since (i) as proven in Theorem 7 the DCR signatures of $\hat{B}$ with challenger $\hat{A}$ remain secure even if the private key of $\hat{A}$ is known, and (ii) the above abort operation is never used in the proof (we added it for ease of presentation and for compliance with the notion of session exposure in the basic security model). Also worth noting is that in the proof the outgoing value X_0 in the test session is provided in the input to $\mathcal{F}$ and then not chosen by $\mathcal{M}$ (thus ensuring that the session is clean). □

[10] It is illustrative to note that some natural key-exchange protocols *do not achieve* this property: Such is the case of key-transport protocols based on public key encryption (such as SSL) or key-exchange protocols that use long term static DH keys g^a, g^b to derive a long-term shared key g^{ab} (see Section 3.1). Examples of protocols that *do provide* resistance to KCI are the the signature-based SIGMA protocol [27], the ISO protocol [11, 22] and the SKEME protocol authenticated via public-key encryption [26].

As shown in Section 3.3, ensuring resistance to KCI attacks requires that the attacker does not learn the DCR signature produced in the protocol which explains the need to define the session key in HMQV as the hash of this signature.

6.2 Perfect Forward Secrecy (PFS)

Another important notion that provides a security guarantee of key-exchange sessions in the case that the attacker has learned the private keys of some parties is Perfect Forward Secrecy (PFS). Informally, a key-exchange protocol is said to have the PFS property if the leakage of the long-term key of a party does not compromise the security of session keys established by that party, and erased from memory, before the leakage occurred. A formal definition of PFS in the context of the model from Section 2 appears at the end of that section. As it turns out, HMQV (nor the original MQV) protocol achieve this property. Moreover, as shown in Section 3.2, no 2-message protocol can achieve PFS (as long as there are no means, such as a secure state shared by the parties in a previous exchange, to authenticate the first message and avoid its replay).

Note that, nonetheless, HMQV does provide a *weak form of forward secrecy*: If the attacker is not actively involved with the choice of the X, Y values at a session (specifically if it does not get to choose or learn the exponents x and y) then the resultant session key does enjoy forward secrecy. While certainly weaker than the usual PFS, this guarantee of forward secrecy is still significant in practice where the mounting of active attacks is harder than simple recording past communications and can usually be performed on a smaller number of sessions. This (unavoidable) weakness of the 2-message HMQV protocol can be solved by adding a third message to the protocol and a key confirmation mechanism as shown in Section 8.

Definition 22 *A key-exchange protocol provides* weak PFS (wPFS) *if an attacker $\mathcal{M}$ cannot distinguish from random the key of any session for which the session and its matching session are clean even if $\mathcal{M}$ has learned the private keys of both peers to the session.*

Note that in the regular definition of PFS (Section 2) one assumes that the session was established at the time that both $\hat{A}$ and $\hat{B}$ were uncorrupted and the attacker only learns the private keys of these parties at a later point after the session key has already expired and erased from memory. In the above formulation, the security guarantee is provided even if the attacker learned the private keys before the session was established, as long as the session (and its matching one) are clean. Of course, this comes at the cost of assuming that the attacker was passive during the session establishment.

Lemma 23 *Under the CDH assumption, protocol HMQV provides weak PFS.*

Proof: We only outline the idea of the proof and omit the details that can be completed following the proof from Section 5.2. Given an attacker $\mathcal{M}$ that breaks the wPFS property, we construct a CDH solver $\mathcal{C}$ as follows. (An important difference between this case and the proof from Section 5.2 is that here we use $\mathcal{M}$ to directly build a CDH solver rather than building a DCR forger.) Let $X = g^x$ and $Y = g^y$ be the inputs to $\mathcal{C}$; the goal of $\mathcal{C}$ is to compute g^{xy}. $\mathcal{C}$ runs $\mathcal{M}$ against a virtual execution of the protocol in which $\mathcal{C}$ chooses all parties' private keys, and also chooses a random guess for the test session against which $\mathcal{M}$ succeeds in its wPFS attack. We call this guessed session the g-session and denote its peers by $\hat{A}$ and $\hat{B}$ (and their private keys by a and b, respectively). $\mathcal{C}$ then uses the input values X, Y as the incoming and outgoing messages in the g-session. If the attacker does choose the g-session as its test session and $\mathcal{M}$ distinguishes its session key from random (both events happen simultaneously with non-negligible probability) then the following must be true. In order to be considered successful $\mathcal{M}$ cannot modify the X, Y values in the test session (otherwise the session is not clean). Hence the established key is $H(\pi_0)$ for $\pi_0 = g^{(x+da)(y+eb)}$, and for distinguishing $H(\pi_0)$ from random $\mathcal{M}$ must output at some point in its run the value π_0. Now, with the knowledge of a and b, $\mathcal{C}$ can compute g^{xy} out of π_0. □

6.3 Reflection Attacks

In the analysis of protocol HMQV we have assumed that the peers $\hat{A}$ and $\hat{B}$ in the test session are different (see Step 3 in the description of forger $\mathcal{F}$ in Section 5.2). In some scenarios, however, a party may want to establish a secure channel with itself (for example, a mobile user that communicates to its desktop computer, while both the mobile device and the desktop have the same identity in the form of the same digital certificate). Here we show how to extend the analysis of Section 5.2 to include such scenarios. In particular, we show that HMQV resists the so called reflection attacks in which the attacker simply copies $\hat{A}$'s authentic messages and sends them back to $\hat{A}$ as the messages coming from the other copy of $\hat{A}$.

The technical difficulty in achieving this result is that we are now concerned with test sessions of the form $(\hat{B}, \hat{B}, X_0, Y_0)$, that is, sessions in which $\hat{A} = \hat{B}$. When we apply the analysis of Section 5.2 to this case we get that if an attacker can successfully win the distinguishing test against the above test session then we can build a forger $\mathcal{F}$ against the dual signature $\mathit{DSIG}_{\hat{B},\hat{B}}(m_1{=}\hat{B}, m_2{=}\hat{B}, X_0, Y_0)$ where X_0 is the value input to $\mathcal{F}$ (as the challenge on which to forge the signature) and Y_0 is chosen by $\mathcal{F}$. The problem, however, is that we have not shown that such forgers do not exist. Indeed, Theorem 7 only guarantees the security of DCR signatures with peers $\hat{A} \neq \hat{B}$. Consequently, we outline here an extension of the proof of Theorem 7 that will allow us to claim the infeasibility of a forger $\mathcal{F}$ as above, and then also the infeasibility of a KE-attacker that wins the distinguishing test.

The difficulty in extending the proof of Theorem 7 to the case $\hat{A} = \hat{B}$ is that the proof of the theorem assumes that the private key a of $\hat{A}$ is known to the forger and to the CDH solver $\mathcal{C}$ (the value of a is required in the computation of W in Equation (1) in order to get rid of the factor A^d in that computation). Here a is unknown since $a = b$ and the latter is secret. Thus, if we apply the same reduction as in Theorem 7 we get a solver $\mathcal{C}$ that computes $(X_0A^d)^b$ rather than X_0^b as needed for deriving $CDH(U, V)$. An "obvious" fix is to change the way $\mathcal{C}$ sets X_0 in Figure 4 from $X_0 = U$ to $X_0 = U/A^d$. This, however, requires knowing the value of $d = \bar{H}(X_0, m_1)$ at the onset of $\mathcal{F}$'s activation which is not necessarily the case in general (indeed, $\mathcal{F}$ can choose the value m_1 that determines d much later in the game). Fortunately, in the case of interest here (that considers a specific signature of the form $\mathit{DSIG}_{\hat{B},\hat{B}}(m_1{=}\hat{B}, m_2{=}\hat{B}, X_0, Y_0)$) the value m_1 is not chosen by $\mathcal{F}$ but rather fixed to $m_1 = \hat{B}$. Thus both m_1 and X_0 are known in advance and thus we can specify $\mathcal{C}$ to set $X_0 = U/A^d$. There is, however, one more obstacle: if the forger $\mathcal{F}$ chooses Y_0 to be X_0 then we get $d = \bar{H}(X_0, \hat{B}) = \bar{H}(Y_0, \hat{B}) = e$, and therefore the value d changes, in the description of $\mathcal{C}$ in Figure 4, before and after the repeat experiment. In this case setting $X_0 = U/A^d$ does not allow computing W.

On the basis of the above discussion we can now go back to the security analysis of protocol HMQV in the case that the attacker $\mathcal{M}$ is assumed to win the distinguishing game with a test session $(\hat{B}, \hat{B}, X_0, Y_0)$. We consider two cases:

Case I: $X_0 \neq Y_0$. In this case the proof from Section 5.2 extends to show the existence of a forger

$\mathcal{F}$ that forges $\hat{B}$'s dual signatures $DSIG_{\hat{B},\hat{B}}(m_1 = \hat{B}, m_2 = \hat{B}, X_0, Y_0)$ for $X_0 \neq Y_0$. Thus, following the above discussion, we build a CDH solver $\mathcal{C}$ that runs the forger $\mathcal{F}$ as in Figure 4 except that $\mathcal{C}$:

1. Chooses a random value d in $\{0,1\}^\ell$ (the range of $\bar{H}$).
2. Sets $X_0 = U/B^d$ and $B = V$ (remember that U, V are the inputs to $\mathcal{C}$ and here $A = B$).
3. Sets $\bar{H}(X_0, m_1)$ to the value d.

The rest of $\mathcal{C}$ is unchanged, including the same calculation of W as in Figure 4 which results in $\mathcal{C}$ solving $CDH(U, V)$.

Case II: $X_0 = Y_0$. In this case a fix of $\mathcal{C}$ as above does not work since, as pointed out, the value of d before and after the repeat experiment changes. To solve this issue we have two approaches. The first is to change the specification of HMQV such that the definition of the values d, e has a "directional indication": for example, define $d = \bar{H}(X, \hat{A}, \text{“0”})$ and $e = \bar{H}(Y, \hat{B}, \text{“1”})$ (here "0" and "1" are used by the initiator and responder of the session, respectively). This guarantees that, even if $X_0 = Y_0$, the value d does not change before and after the repeat experiment, and therefore the argument in Case I holds here too, showing that the protocol is secure in the case $X_0 = Y_0$ too.

The second approach shows that the above modification to HMQV is not necessary. For this, observe that when $X_0 = Y_0$ then these values are chosen by the uncorrupted $\hat{B}$ and therefore we are in a situation similar to that of Section 6.2 in which the Diffie-Hellman values are chosen in "clean sessions". Specifically, by applying the argument in the proof of Lemma 23 one can show how to transform an attacker that can distinguish the key of session $(\hat{B}, \hat{B}, X_0, X_0)$ from random into a CDH solver. In this case, however, we can only guarantee this solver to work on inputs X, Y with $X = Y$. In other words, we can build an algorithm that given g^{x_0} computes $g^{x_0^2}$. But, as observed by Maurer and Wolf [31], such an algorithm can in turn be used to solve the general CDH problem (namely, where the inputs are arbitrary values X, Y in G).

Summarizing the above we have the following lemma:

Lemma 24 *Under the CDH assumption, sessions established in the HMQV protocol between an uncorrupted peer and itself are secure. In particular, successful reflection attacks are not feasible.*

We note that this lemma together with Lemmas 9 and 10 completes the proof of Theorem 8.

7 HCR Signatures and Resilience to Leakage of DH Exponents

As we have stated repeatedly, a main security requirement for key-exchange protocols is to confine the damage caused by the leakage of session-specific ephemeral information to the compromised session only (and its matching session, if it exists). In particular, Diffie-Hellman protocols should ensure that the leakage of ephemeral DH exponents does not affect the security of sessions where this exponent is not used. A good illustration of a protocol *not* having this property is the signature-based 2-message protocol discussed at the beginning of Section 3.1. In that case, stealing the value of a DH exponent from a single session between $\hat{A}$ and $\hat{B}$ suffices for the attacker to impersonate $\hat{A}$ to $\hat{B}$ forever, and without having to learn anything about $\hat{A}$'s or $\hat{B}$'s private keys.

The present section is devoted to showing the robustness of the HMQV protocol against such ephemeral leakage of Diffie-Hellman exponents. We show that disclosing to the attacker the exponent used in an HMQV session has no bearing on the security of other sessions. Formally, we show that even if the ephemeral session DH exponents are stored in the session's state, the protocol remains secure in the face of state-reveal queries through which the attacker learns these exponents. Moreover, resilience to the leakage of ephemeral DH exponents is important to protect a party that pre-computes pairs (y, g^y) for use at later sessions (pre-computation reduces the on-line cost of the protocol to 1.5 exponentiations). Discovering such a pair helps the attacker in two ways: learning y and being able to predict which value of g^y will be used in a session. We show that the best protection against such an attacker is obtained by adding the parties' identities to the key derivation hashing or by adding an unpredictable nonce to the protocol messages.

A significant technical point that is worth noting is that resilience to the disclosure of ephemeral DH exponents does not follow from the analysis in Sections 5 and 6. That analysis refers to the formal specification of HMQV in Section 5.1 which did not allow for the storage of ephemeral exponents in the session state (thus requiring that ephemeral exponents be as protected as the long term private key). Moreover, the fact that the attacker is not allowed to see these ephemeral exponents is used in an essential way in that analysis. This is so since the proof of unforgeability of XCR signatures (in Section 4.2), on which the security of HMQV relies, does not carry on to the case in which the attacker learns an ephemeral value y used by the signer $\hat{B}$ in one of its signatures. Indeed, observe that in the simulation of $\hat{B}$'s signatures performed by the CDH solver $\mathcal{C}$ in Figure 4 (specifically steps S1-S3) the simulator does not know the value y (and thus cannot present it to $\mathcal{F}$ if such a value is queried). Consequently, the analysis of HMQV in Section 5.2 that builds on the security of XCR signatures does not apply to the case where the ephemeral DH exponents are revealed.

To overcome this problem we are going to consider in this section a "hashed variant" of XCR signatures (which we call HCR signatures) that we prove to be unforgeable even if the exponents y used by the signer in its signatures are provided to the forger. On the basis of these signatures we are able to claim the security of HMQV even in the face of leaked DH exponents. This, however, requires replacing the CDH assumption used in our previous analysis with two substantially stronger assumptions that we recall below: Gap Diffie-Hellman and KEA1. Yet, we emphasize that these stronger assumptions are not needed for any one of the other properties proven to hold for HMQV (this is the main reason to treat the resilience to DH exponents separately).

Definition 25 [37] *Let G be a cyclic group generated by an element g. 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 (X, Y, Z), for $X, Y \in G$, oracle $\mathcal{O}$ outputs 1 if and only if $Z = CDH_g(X, Y)$. 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.*

What this assumption says is that in the group G computing $CDH(X, Y)$, for $X, Y \in_{\mathrm{R}} G$, is strictly harder than deciding if a given Z equals $CDH(X, Y)$.

The following assumption was first stated and used in [14]; it was re-defined and further studied in [18, 4]. The name KEA1 (where KEA stands "knowledge of exponent assumption") is from [4]. The statement of the assumption presented next is somewhat informal; please refer to the above works for the details. The KEA1 assumption captures the intuition related to the difficulty of the Diffie-Hellman problem by which the "only way" to produce a pair of the form (C, C^b) when all is given is a pair (g, g^b), where g is a generator of a group of prime order q, is to choose $c \in Z_q$ and compute $(g^c, (g^b)^c)$.

Definition 26 [14, 18, 4] *Let G be a cyclic group of prime order q generated by an element g, and consider algorithms that on input a pair (g, g^b) output a pair (C, D) in G^2. Such an algorithm E is* *said to be a KEA1 algorithm if with non-negligible probability (over the choice of b and E's random coins) $E(g, g^b)$ outputs (C, D) such that $D = C^b$. We say that* the KEA1 assumption holds *over G if for every efficient KEA1 algorithm E in G there exists another efficient algorithm E' for which the following property holds except for a negligible probability: let (g, g^b) be an input to E and ρ a vector of random coins for E on which E outputs $(C, D = C^b)$ then on the same inputs (and random coins) E' outputs the triple $(C, D = C^b, c)$ where $C = g^c$.*

The rest of the section is organized as follows: Section 7.1 introduces HCR signatures and their security requirements; Section 7.2 proves the unforgeability of these signatures under the above assumptions; Section 7.3 deals with the case in which a DH value Y used in a signature is known in advance by the forger. Finally, Section 7.4 uses these results to prove the resilience of HMQV to the leakage of ephemeral DH exponents.

7.1 HCR Signatures

Here we introduce Hashed XCR signatures (HCR, for short) defined exactly as XCR signatures except that instead of a pair (Y, σ) as output by regular XCR signatures (on input message m and challenge X), the HCR signature outputs $(Y, H(\sigma))$ where H outputs k bits (same output length as used in producing the key in the HMQV protocol). The computation of σ by a signer $\hat{B}$ with public key $B = g^b$ is exactly as in XCR, namely, $\sigma = \mathit{XSIG}(Y, m, X) = X^{y+eb}$, where $e = \bar{H}(Y, m)$ is of length ℓ bits. We denote the modified signature by HSIG, i.e., $\mathit{HSIG}(Y, m, X) = H(X^{y+eb})$. Note that in spite of the hashing of σ such a signature can be verified by the challenger that knows the dlog of X; in particular, the HMQV protocol uses such signatures. Our main motivation to consider HCR signatures (in addition to XCR ones) is that they enjoy the property of being *unforgeable even if the attacker knows the ephemeral exponent y corresponding to $Y = g^y$*.[11]

In order to formalize this property in a way that is applicable to the security analysis of HMQV we modify the forgery game from Figure 3 and present the modified game in Figure 5. In this new game we are making the task of the forger $\mathcal{F}$ harder by providing it with only a hashed version of XCR signatures, but at the same time making it easier for $\mathcal{F}$ by disclosing to it the exponent y used by $\hat{B}$ to compute each signature component $Y = g^y$. Moreover, we will accept a forgery $(Y_0, m_0, \mathit{HSIG}(Y_0, m_0, X_0)$ as valid even if $\mathcal{F}$ is unable to compute $y_0 = \log_g(Y_0)$. We also let $\mathcal{F}$ choose its challenge value X presented to the signer only after the signer disclosed its value y for the signature. As we pointed out in Remark 4.3, this modified order of interaction captures the interaction induced by the HMQV protocol.

7.2 Unforgeability of HCR signatures

Lemma 27 *Under the GDH and KEA1 assumptions, the HCR signature scheme is secure in the random oracle model, namely, no efficient forger can succeed with non-negligible probability in the forgery game of Figure 5.*

Proof: Let $\mathcal{F}$ be a forger against HCR signatures that wins with non-negligible probability the forgery game of Figure 5. We will use such $\mathcal{F}$ to build a successful CDH solver $\mathcal{C}$ which is equipped with a DDH oracle, thus contradicting the Gap Diffie-Hellman assumption. The general structure of $\mathcal{C}$ is similar to the CDH solver built in the proof of Theorem 5 and presented in Figure 4, yet it

[11] See Remark 4.5 for a comparison of HCR and DSS signatures.

1. Forger $\mathcal{F}$ is given values B, X_0 where $B, X_0 \in_{\mathrm{R}} G$.
2. $\mathcal{F}$ is given access to a signing oracle $\hat{B}$ (representing an HCR signer $\hat{B}$ with private key b and public key $B = g^b$).
3. Each signature query from $\mathcal{F}$ to $\hat{B}$ consists of the following interaction
   (a) $\mathcal{F}$ presents $\hat{B}$ with a message m;
   (b) $\hat{B}$ responds with $y \in_{\mathrm{R}} Z_q$ and stores the triple $(y, Y = g^y, m)$ as an "incomplete triple"
   (c) $\mathcal{F}$ presents $\hat{B}$ with a pair (Y, m) and a challenge X.
   (d) $\hat{B}$ checks that the pair (Y, m) is in one of its incomplete triples and that $X \neq 0$; if not, it aborts, else it responds with $r = \mathit{HSIG}_{\hat{B}}(Y = g^y, m, X) = H(X^{y+\bar{H}(Y,m)b})$, marks the triple (y, Y, m) as complete and stores with it the signature values $X, e = \bar{H}(Y, m), r$.
4. $\mathcal{F}$ is allowed a polynomial number of signature queries (as in 3) to $\hat{B}$ where the queries are chosen (possibly adaptively) by $\mathcal{F}$. We allow for the interaction steps from different signature queries may be interleaved. (This is why $\hat{B}$ records incomplete and complete triples.)
5. $\mathcal{F}$ halts with output "fail" or with a *guess* in the form of a triple (Y_0, m_0, r).

$\mathcal{F}$'s guess is called a *successful forgery* if the following two conditions hold:

   (a) The pair (Y_0, m_0) did not appear in any of the responses of $\hat{B}$ to $\mathcal{F}$'s queries and $Y_0 \neq 0$.
   (b) $r = \mathit{HSIG}_{\hat{B}}(Y_0, m_0, X_0)$ (where X_0 is the value provided as input to $\mathcal{F}$).

We say that $\mathcal{F}$ wins the game (or simply *forges*) if it outputs a successful forgery.

Figure 5: Forgery Game for HCR Signatures

has some important differences resulting from a different way to simulate the signer's answers. In particular, in the present case $\mathcal{C}$ does not need to simulate the computation of the signature value σ but just return a random value for $H(\sigma)$. The problem is how to provide consistent answers to the forger $\mathcal{F}$. For example, if $\mathcal{C}$ returns a random value r as the value of $H(\sigma)$, where $\mathcal{C}$ does not know the actual value of σ, and later $\mathcal{F}$ queries H on the explicit value σ (which $\mathcal{F}$ may be able to compute if it chose the challenge x) how does $\mathcal{C}$ know to return the same value r? It is for this purpose that we need to use a DDH oracle (which is also why we need to assume the Gap DH assumption). The full description of $\mathcal{C}$ is presented in Figure 6.

We now analyze the behavior of $\mathcal{C}$. First note that the simulation of the signer $\hat{B}$ by $\mathcal{C}$ is perfect except for the negligible probability that the input $(Y = g^y, m)$ was queried from $\bar{H}$ before $\mathcal{C}$ chose Y. However, since this choice by $\mathcal{C}$ is done independent of previous computations, then the probability that (Y, m) was queried earlier is at most Q/q where Q is the number of $\bar{H}$-queries in the whole simulation. It is also worth noting the essential use of the DDH oracle needed to check whether the value of a signature $\sigma = X^{y+be}$ was previously presented as input to the oracle H. This is required since if $\mathcal{F}$ chose the challenge $X = g^x$ knowing x then it can compute the actual value of σ and therefore it can query H on σ. For the simulation to succeed, $\mathcal{C}$ must make sure that it answers the H query and the simulated output of the HCR signature consistently. The same argument holds for the consistency check of other H-queries.

Building a CDH solver $\mathcal{C}$ from an HCR forger $\mathcal{F}$

Setup. Given a successful HCR-forger $\mathcal{F}$ we build an algorithm $\mathcal{C}$ that having access to a DDH oracle solves the CDH problem over G. The inputs to $\mathcal{C}$ are a generator g of G and a pair $U = g^u, V = g^v \in_{\mathrm{R}} G$. The goal of $\mathcal{C}$ is to compute $CDH(U, V) = g^{uv}$.

Initializing $\mathcal{F}$. $\mathcal{C}$ sets $B = V$ and $X_0 = U$, and runs the forger $\mathcal{F}$ on input (B, X_0) against a simulated signing oracle, denoted $\hat{B}$, under public key B. $\mathcal{C}$ provides $\mathcal{F}$ with a random tape and also provides answers to the random oracle queries $\bar{H}$ and H generated in the run as specified below.

Simulation of signer $\hat{B}$. The interaction between $\mathcal{F}$ and $\hat{B}$ takes four steps as specified in Figure 5. The actions of $\hat{B}$ in this interaction are simulated by $\mathcal{C}$ as follows.

S1. $\mathcal{F}$ presents to $\hat{B}$ a message m for signing.

S2. $\mathcal{C}$ chooses $y \in_{\mathrm{R}} Z_q$ and $e \in_{\mathrm{R}} \{0,1\}^{\ell}$, and sets $\bar{H}(Y, m)$ to e (if (Y, m) was queried earlier from $\bar{H}$ then $\mathcal{C}$ aborts). $\mathcal{C}$ responds to $\mathcal{F}$, on behalf of $\hat{B}$, with y and e. It stores the values e and the triple (y, Y, m) which it marks as an "incomplete triple".

S3. $\mathcal{F}$ presents $\hat{B}$ with a pair (Y, m) and a challenge X.

S4. $\mathcal{C}$ checks that the pair (Y, m) is part of one of its incomplete triples and that $X \neq 0$; if not the query is ignored. Else $\mathcal{C}$ checks for every value $\sigma \in G$ previously used as input to the oracle H whether $\sigma = X^{y+be}$: it does so using the DDH oracle, namely, by checking whether $CDH(X, B) = (\sigma/X^y)^{1/e}$. If the answer is positive then $\mathcal{C}$ sets r to the already determined value of $H(\sigma)$, otherwise r is set to a random value in $\{0,1\}^k$; $\mathcal{C}$ marks the triple (y, Y, m) as complete, stores a vector (y, m, e, X, r), and returns r to $\mathcal{F}$.

Simulation of H queries. Queries of $\mathcal{F}$ to the random oracle are answered by $\mathcal{C}$ as follows. If the same query was made earlier then $\mathcal{C}$ responds with the same original answer. New queries to $\bar{H}$ are answered with a random value in $\{0,1\}^{\ell}$. If a new input σ is queried from H, then $\mathcal{C}$ checks whether $\sigma = X^{y+be}$ for any one of the vectors (y, m, e, X, r) stored by $\mathcal{C}$ (as before this check is done using the DDH oracle). If equality holds then the corresponding r is returned as $H(\sigma)$, else a random r in $\{0,1\}^k$ is returned.

Upon $\mathcal{F}$'s termination. When $\mathcal{F}$ halts with a forgery guess Y_0, m_0, r_0 (i.e., r_0 is a guess by $\mathcal{F}$ for $HSIG_{\hat{B}}(Y_0, m_0, X_0)$), $\mathcal{C}$ checks whether:

F1. The pair (Y_0, m_0) is valid, i.e. it was not used as the (Y, m) pair in any of the signatures generated by $\hat{B}$ in steps S1-S4 and also $Y_0 \neq 0$.

F2. The value $\bar{H}(Y_0, m_0)$ was queried from the random oracle; if so, we denote the response of $\bar{H}$ to this query by e_0.

F3. The value r_0 was set earlier as $H(\sigma_0)$ for some queried value σ_0.

If these conditions hold, $\mathcal{C}$ proceeds to run the following "repeat experiment", else it aborts.

The repeat experiment. $\mathcal{C}$ runs $\mathcal{F}$ again for a second time under the same input (B, X_0) and using the same coins for both $\mathcal{C}$ and $\mathcal{F}$. The difference between the two runs is in the way in which $\mathcal{C}$ answers the $\bar{H}$ queries during the second run. Specifically, all queries to $\bar{H}$ performed before the $\bar{H}(Y_0, m_0)$ query are answered identically as in the first run. The query $\bar{H}(Y_0, m_0)$, however, is answered with a new independent value $e'_0 \in_{\mathrm{R}} \{0,1\}^{\ell}$. Subsequent queries to $\bar{H}$ are also answered at random from $\{0,1\}^{\ell}$, independently of the responses provided in the first run.

C's Output. If at the end of the second run, $\mathcal{F}$ outputs a forgery guess (Y_0, m_0, r'_0) (with the same (Y_0, m_0) as in the first run), then $\mathcal{C}$ checks that conditions $F1 - F3$ hold wrt (Y_0, m_0, r'_0). If not $\mathcal{C}$ aborts. Else, $\mathcal{C}$ sets σ'_0 to be the value in which $H(\sigma'_0) = r'_0$, it computes $W = (\sigma_0/\sigma'_0)^{(e_0 - e'_0)^{-1}}$, and outputs W as a guess for g^{uv}.

Figure 6: Proof of Lemma 27

As a result of this almost perfect simulation of $\hat{B}$ in its game with $\mathcal{F}$, we see that the probability that $\mathcal{F}$ outputs a correct forgery in its run by $\mathcal{C}$ is, up to a negligible difference, the same as in a real run of $\mathcal{F}$, and hence non-negligible. Below we will argue that whenever $\mathcal{F}$ outputs a correct forgery Y_0, m_0, r_0 under the run by $\mathcal{C}$ then the three conditions F1-F3 checked by $\mathcal{C}$ hold except for a non-negligible probability. Therefore, except for such a negligible probability, a successful run of $\mathcal{F}$ (i.e., in which the triple Y_0, m_0, r_0 is a valid forgery wrt X_0) will produce a value σ_0 as in condition F3 and such that $\sigma_0 = X_0^{y_0+e_0b}$. Since $\mathcal{F}$ outputs correct forgeries with non-negligible probability then we have that with non-negligible probability the first run of $\mathcal{F}$ ends with a value $\sigma_0 = X_0^{y_0+e_0b}$. In this case, the repeat experiment will be performed. By (a straightforward adaptation of) the Forking Lemma [38] we get that with non-negligible probability this second run ends with a forgery Y_0', m_0', r_0' where $(Y_0', m_0') = (Y_0, m_0)$ and $r_0' = H(\sigma_0')$ where σ_0' is a value presented by $\mathcal{F}$ as an input to H and also $\sigma_0' = X_0^{y_0+e_0'b}$. In this case the value $(\sigma_0/\sigma_0')^{(e_0-e_0')^{-1}}$ is equal to X_0^b which in turn is equal to g^{uv}. In all, the probability that $\mathcal{C}$ outputs a correct value $CDH(U, V)$ is non-negligible.

All is left to prove is that in the case in which the forgery Y_0, m_0, r_0 output by $\mathcal{F}$ is correct and valid then conditions F1-F3 checked by $\mathcal{C}$ hold except for a negligible probability. This is obvious for F1 since this is a condition for the forgery to be valid. In the case of F2, if $\bar{H}(Y_0, m_0)$ was not queried then the probability for $\mathcal{F}$ to output a correct forgery is at most $1/2^\ell + 1/2^k$: for each value of $e_0 = \bar{H}(Y_0, m_0)$ the induced value $\sigma_0 = X_0^{y_0+e_0b}$ is different and then hitting the right r_0 requires guessing e_0 or guessing $H(\sigma_0)$ without knowing σ_0. In the case of condition F3, guessing r_0 without querying $H(\sigma_0)$ has only a probability of $1/2^k$ to succeed. However, the value r_0 may be chosen by $\mathcal{F}$ as one of the random values r answered by $\mathcal{C}$ (on behalf of $\hat{B}$) in the simulation step S4, say an r taken from a vector (y, m, e, X, r) stored by $\mathcal{C}$. In this case, even though neither $\mathcal{C}$ nor $\mathcal{F}$ queried H on the value $\sigma = X^{y+eb}$ (nor on $\sigma_0 = X_0^{y_0+e_0b}$), the forgery by $\mathcal{F}$ would be correct if it happens that

$$X_0^{y_0+e_0b} = X^{y+eb} \tag{2}$$

where $e_0 = \bar{H}(Y_0, m_0)$, $e = \bar{H}(Y, m)$, X_0 is the input to $\mathcal{F}$, and X the challenge (chosen by $\mathcal{F}$) under which $\hat{B}$ produced the signature (y, m, r). However, if this is the case then we cannot use algorithm $\mathcal{C}$ from Figure 6 to compute the DH value X_0^b since in order to do that $\mathcal{C}$ needs to learn the value σ_0. What we are going to show, in Lemma 28 below, is that the probability that algorithm $\mathcal{F}$ finds a forgery that satisfies Equation (2) is negligible. This will end the proof of Lemma 27. □

In order to formalize and prove Lemma 28 below, we first define a class of forgers $\mathcal{F}$, which we call collision forgers, that interact with an HCR-signer $\hat{B}$ with public key B as specified in Step 3 of Figure 5. Without loss of generality, we will assume that $\mathcal{F}$ either aborts or else it outputs a signature triple (Y_0, m_0, r_0) as a candidate forgery; we also assume that $r_0 \neq H(\sigma)$ for all values of σ explicitly queried from H (we are only interested on the ability of $\mathcal{F}$ to output triples (Y_0, m_0, r_0) that violate condition F3). We will call such a triple (Y_0, m_0, r_0) a collision forgery, and say it is *correct* if (i) the pair (Y_0, m_0) is valid (i.e. $Y_0 \neq 0$ and (Y_0, m_0) was not used in any of the signatures issued by $\hat{B}$), (ii) $\hat{B}$ generated a signature (y, m, r) under some challenge X (chosen by $\mathcal{F}$) for which $r = r_0$, and (iii) Equation (2) holds. In this case we call (Y_0, m_0, r_0) the collision forgery and call the triple (Y, m, r) the colliding signature (of $\hat{B}$ wrt some challenge X).

Lemma 28 *Under the Gap Diffie-Hellman and KEA1 assumptions, there is no efficient collision forger against HCR signatures in the random oracle model.*

Proof: We assume that there is an efficient algorithm $\mathcal{F}$ that contradicts the lemma's statement. Specifically, we will assume that $\mathcal{F}$ interacts with a signer $\hat{B}$ (with public key B) and eventually outputs either "abort" or a collision forgery (Y_0, m_0, r_0) such that for some signature (y, m, r) issued by $\hat{B}$ with respect to a challenge X (chosen by $\mathcal{F}$) it holds that $r = r_0$ (and r was not output by H on any explicit query σ); in this case we call (y, m, r) the colliding signature. For contradiction, we assume that with non-negligible probability $\mathcal{F}$ outputs a correct collision forgery.

We prove the lemma by considering two cases. The first assumes that with non-negligible probability $\mathcal{F}$ solves Equation (2) with a value of Y_0 (in the collision forgery) different than Y (in the colliding signature); the second case assumes that with non-negligible probability $\mathcal{F}$ succeeds with $Y_0 = Y$ (obviously, at least one of the two cases must hold with non-negligible probability).

Case I: $Y_0 \neq Y$**.** We show how to build from $\mathcal{F}$ an algorithm $\mathcal{D}$ that solves the discrete logarithm problem over G. We start by defining an algorithm $\mathcal{C}$ similar to the one described in Figure 6 but with some important changes.

$\mathcal{C}$ runs $\mathcal{F}$ as in Figure 6, in particular by simulating $\hat{B}$ to $\mathcal{F}$ via the simulation steps S1-S4. However, before this simulation starts, $\mathcal{C}$ chooses $i \in_{\rm R} \{1, \ldots, n\}$, where n is a bound on the number of signature queries performed by $\mathcal{F}$, and three other values $y_1 \in_{\rm R} Z_q$, $e_1 \in_{\rm R} \{0,1\}^\ell$ and $r_1 \in_{\rm R} \{0,1\}^k$. Simulation of signature queries is identical to Figure 6 except that when $\mathcal{F}$ asks its i-th signature query, say on message m_1, $\mathcal{C}$ uses y_1 as the y-value of this signature. In this case, $\mathcal{C}$ also sets $\bar{H}(Y_1 = g^{y_1}, m_1)$ to the value e_1 (except if $\bar{H}(Y_1, m_1)$ was queried before in which case $\mathcal{C}$ aborts). When presented with the challenge X for this signature, $\mathcal{C}$ sets the r-value of the signature to r_1; we denote the challenge X presented by $\mathcal{F}$ in this query by X_1.

If $\mathcal{F}$ outputs a forgery guess (Y_0, m_0, r_0) where $r_0 = r_1$ (i.e., an alleged forgery collision with the i-th signature by $\hat{B}$) and (Y_0, m_0) is valid (i.e., not used by $\hat{B}$ in any of its signatures and with $Y_0 \neq 0$) then $\mathcal{C}$ performs the following "repeat experiment"; in all other cases $\mathcal{C}$ aborts its run.

The repeat experiment: $\mathcal{C}$ rewinds its computation up to the point in which $\mathcal{F}$ queried the value $\bar{H}(Y_0, m_0)$ and re-runs $\mathcal{F}$ from this point on but this time with new random values for the $\bar{H}(Y_0, m_0)$ query and subsequent $\bar{H}$-queries. We denote by e_0 the response to $\bar{H}(Y_0, m_0)$ in the first run and by e_0' the response to $\bar{H}(Y_0, m_0)$ during the repeat experiment. In addition, $\mathcal{C}$ makes the following modification to its re-run of $\mathcal{F}$: if in the first run of $\mathcal{F}$ the simulation used the pair (y_1, e_1) *after* the $\bar{H}(Y_0, m_0)$ query took place, then in the repeat experiment $\mathcal{C}$ chooses at random (among the remaining sessions, and independently of the choice of i in the first run) the signature query in which to embed the values y_1, e_1, r_1. If the $\bar{H}(Y_0, m_0)$ query happened after the pair (y_1, e_1) was used by $\mathcal{C}$ then the above modification is not needed.

If both the first and second run of $\mathcal{F}$ by $\mathcal{C}$ end up with valid collision forgeries with the same pair (Y_0, m_0) and the same values (y_1, e_1, r_1) in the colliding signature, then we denote by m_1, X_1 the message and challenge used in the colliding signature in the first run, and by m_1', X_1' the corresponding values from the colliding signature in the second run. In this case, $\mathcal{C}$ outputs the values: $Y_0, m_0, e_0, y_1, e_1, m_1, m_1', X_1 \neq 0, X_1' \neq 0$ as well as two values C_1, C_2 computed as:

$$C_1 = X_0^{e_0 - e_0'} \left(\frac{X_1}{X_1'}\right)^{-e_1} \quad \text{and} \quad C_2 = \left(\frac{X_1}{X_1'}\right)^{y_1} \tag{3}$$

In all other cases $\mathcal{C}$ aborts.

We proceed to analyze $\mathcal{C}$. By assumption, when interacting with a real signer $\hat{B}$, $\mathcal{F}$ outputs a correct collision forgery with non-negligible probability. Since the simulation of $\hat{B}$ by $\mathcal{C}$ is perfect up to a negligible probability (see the proof of Lemma 27) then the non-negligible probability of success of $\mathcal{F}$ holds too when $\mathcal{F}$ is run by $\mathcal{C}$. Also, with non-negligible probability $\mathcal{C}$ guesses correctly the signature query in which the colliding signature is computed. Therefore with non-negligible probability $\mathcal{F}$ ends its first run by $\mathcal{C}$ with a correct collision forgery (wrt the challenge X_0) with parameters denoted as Y_0, m_0, e_0, r_0 and with a colliding signature with parameters y_1, m_1, e_1, r_1, X_1 where (y_1, e_1, r_1) were chosen by $\mathcal{C}$ at the beginning of its run and m_1, X_1 were chosen by $\mathcal{F}$. Using (an adaptation of) the Forking Lemma from [38] one gets that all of the above holds for the second run of $\mathcal{F}$ under the repeat experiment (which also has a non-negligible probability of guessing the correct i in which to embed the values y_1, e_1, r_1). More precisely, there is a non-negligible probability that $\mathcal{F}$ will output in both first and second runs a correct collision forgery with the *same* parameters Y_0, m_0, r_0, and the same colliding values $y_1, e_1, r_1 = r_0$, but possibly different values of m_1 and X_1 (since the choice of these elements by $\mathcal{F}$ may depend on the response to the $\bar{H}(Y_0, m_0)$ query). Hence, we let m_1, X_1 denote the corresponding values from the first run of $\mathcal{F}$, and m_1', X_1' denote the values from the second run. We note that even if $m_1 \neq m_1'$, we have by construction of $\mathcal{C}$ that the values $\bar{H}(Y, m_1)$ and $\bar{H}(Y, m_1')$ are both equal to e_1.

Putting all together, we have that with non-negligible probability C outputs a sequence of values $Y_0, m_0, e_0, y_1, e_1, m_1, m_1', X_1.X_1'$ which satisfy Equation (2), namely,

$$X_0^{y_0+e_0b} = X_1^{y_1+e_1b}$$
$$X_0^{y_0+e_0'b} = X_1'^{y_1+e_1b}$$

Re-arranging these two equations and dividing one by the other we obtain that

$$\left(X_0^{e_0-e_0'}\left(\frac{X_1}{X_1'}\right)^{-e_1}\right)^b = \left(\frac{X_1}{X_1'}\right)^{y_1} \tag{4}$$

In other words, we see that the values C_1 and C_2 output by $\mathcal{C}$ satisfy that $C_1^b = C_2$. By the KEA1 assumption there exists an algorithm $\mathcal{C}'$ that in addition to all values output by $\mathcal{C}$ also outputs a value t such that $t = \log_g(C_1) = (e_0 - e_0')x_0 - e_1\Delta x_1$ where $x_0 = \log_g(X_0)$ and $\Delta x_1 = \log_g(X_1/X_1')$. Note that by the equality $C_1^b = C_2$ we also have that t satisfies $bt = y_1\Delta x_1$.

We are now ready to describe a dlog solver $\mathcal{D}$. Algorithm $\mathcal{D}$ receives as input a value $X_0 \in_{\mathrm{R}} G$. It chooses an element $b \in_{\mathrm{R}} Z_q$, sets $B = g^b$, and runs $\mathcal{C}'$ on inputs $B = g^b, X_0$. When $\mathcal{C}'$ produces its output, including the value t, then $\mathcal{D}$ sets the two equations induced by t, namely:

$$t = (e_0 - e_0')x_0 - e_1\Delta x_1 \quad \text{and} \quad bt = y_1\Delta x_1. \tag{5}$$

Since $\mathcal{D}$ knows b and t, these represent two linear equations on two unknowns x_0 and Δx_1, which can be solved to obtain X_0. Note that these equations have a unique solution provided that $e_0 \neq e_0'$ which is the case except for probability $2^{-\ell}$.

Summarizing, we have shown that if $\mathcal{F}$ is an efficient collision forger that succeeds with non-negligible probability then we can build an efficient algorithm $\mathcal{D}$ that solves the discrete logarithm problem over the group G. This ends the proof of Case I.

Case II: $Y_0 = Y$. Since in the proof of case I we have used the notation Y_1 instead of Y we will keep this notation here, i.e., we assume that the component Y_0 in the forgery output by $\mathcal{F}$ equals the value $Y_1 = g^{y_1}$ generated by $\mathcal{C}$. In the analysis of case I we have used the fact that $\bar{H}(Y_0, m_0)$ and $\bar{H}(Y_1, m_1)$ are chosen independently of each other so that a change in the response to $\bar{H}(Y_0, m_0)$ did not result in changing e_1. This is not the case, however, when $Y_0 = Y_1$; in this case the whole argument used in Case I collapses. Fortunately, in Case II we can take advantage of the fact that $Y_0 = Y_1$ in a different way: now, we know also the exponent y_0 for which $Y_0 = g^{y_0}$ since $y_0 = y_1$ and the latter is chosen by $\mathcal{C}$.

Using the above fact, we define a new algorithm $\mathcal{C}$ that on input $X_0, \hat{B} \in_{\mathrm{R}} G$, interacts with the forger $\mathcal{F}$ as in Figure 6. This time, however, we do not need $\mathcal{C}$ to choose in advance the values y_1, e_1 for the colliding signature nor it is needed to guess which query from $\hat{B}$ produces this colliding signature. Moreover, we can get rid of the repeat experiment all together. All is needed is to specify that when $\mathcal{F}$ outputs a collision forgery (Y_0, m_0, r_0) wrt challenge X_0 which collides with a signature (y_1, m_1, r_1) output by $\hat{B}$ under challenge X_1 (i.e., $r_1 = r_0$) and such that $Y_0 = Y_1$, then $\mathcal{C}$ outputs the sequence of values: $Y_0, m_0, e_0 = \bar{H}(Y_0, m_0), y_1 = y_0, e_1 = \bar{H}(Y_1, m_1), m_1, X_1$ as well as two values C_1, C_2 computed as $C_1 = X_0^{e_0} X_1^{e_1}$ and $C_2 = (X_1/X_0)^{y_1}$.

Analyzing the behavior of $\mathcal{C}$ we see that, if in a run under $\mathcal{C}$, the forger $\mathcal{F}$ outputs a correct collision forgery (Y_0, m_0) which collides with a signature (y_1, m_1) output by $\hat{B}$ (and with values $X_0, X_1, y_0 = y_1, m_1, y_1, e_0, e_1$ as specified above) then by virtue of Equation (2) and since $Y_0 = Y_1$ we have that $X_0^{y_1 + e_0 b} = X_1^{y_1 + e_1 b}$. Re-arranging terms and denoting $C_1 = X_0^{e_0} X_1^{-e_1}$ and $C_2 = (X_1/X_0)^{y_1}$, we have that in the case of a correct collision forgery by $\mathcal{F}$, $\mathcal{C}$ succeeds in finding two (known) constant C_1, C_2 such that $C_1^b = C_2$. Thus, using the KEA1 assumption we know that there is an algorithm $\mathcal{C}'$ that, in addition to the other outputs of $\mathcal{C}$, it also outputs $t = \log_g(C_1) = e_0 x_0 - e_1 x_1$ (where $x_0 = \log_g(X_0)$ and $x_1 = \log_g(X_1)$).

We are now ready to define an algorithm $\mathcal{D}$ for solving the dlog problem for input $X_0 \in_{\mathrm{R}} G$. This algorithm $\mathcal{D}$ chooses $b \in_{\mathrm{R}} Z_q$ and runs C' on inputs $B = g^b$ and X_0. When $\mathcal{C}'$ halts with a (non-abort) output as above then $\mathcal{D}$ can set two linear equations $t = e_0 x_0 - e_1 x_1$ and $bt = y_1 x_0 - y_1 x_1$ on unknowns x_0, x_1. In the case that $y_1 \neq 0$ and $e_0 \neq e_1$, these equations have a unique solution from which $\mathcal{D}$ can obtain $x_0 = \log_g(X_0)$. Hence $\mathcal{D}$ successfully computes the dlog of its input X_0 whenever the following three conditions are satisfied: (i) the run of $\mathcal{F}$ under $\mathcal{C}$ produces a correct collision forgery, (ii) $y_1 \neq 0$, and (iii) $e_0 = \bar{H}(Y_0, m_0) \neq e_1 = \bar{H}(Y_1, m_1)$. The first happens with non-negligible probability by assumption on $\mathcal{F}$ and by the almost-perfect simulation of $\mathcal{C}$. Condition (ii) happens with probability $1 - 1/q$ since y_1 is chosen by $\mathcal{C}$ (or $\hat{B}$) at random in Z_q. The last condition ($e_0 \neq e_1$) happens as long as $\bar{H}(Y_0, m_0) \neq \bar{H}(Y_1, m_1)$. Now, since $m_0 \neq m_1$ (otherwise, since $Y_0 = Y_1$ we would have $(Y_0, m_0) = (Y_1, m_1)$ which would make $\mathcal{F}$'s forgery invalid), then the probability that $\bar{H}(Y_0, m_0) = \bar{H}(Y_1, m_1)$ is $2^{-\ell}$. □

This completes the proof of Lemma 28 and with it the proof of Lemma 27.

7.3 HCR Signatures with Off-Line Computed Y

In Section 7.2 we proved that if the exponent y of the $Y = g^y$ component in an HCR signature generated by the legitimate signer is revealed then the scheme still remains unforgeable. However, the analysis there assumed that the value y is chosen randomly at the time of the signature generation (i.e., in Step 3b of Figure 5) and only then y is revealed. Moreover, the analysis used the fact that $Y = g^y$ is unpredictable to the forger before this value is output by the signer $\hat{B}$. This is used in an essential way in the proof of Lemma 28 when arguing that in case II the probability that $\bar{H}(Y, m_0) = \bar{H}(Y, m)$ for $m \neq m_0$ is $2^{-\ell}$ and, similarly, that in case I the probability $\bar{H}(Y_0, m_0) = \bar{H}(Y_1, m_1)$ where $Y_0 \neq Y_1$ (and m_0, m_1 unrestricted) is also at most $2^{-\ell}$. In the case that the attacker knows Y in advance the analysis needs to consider that the attacker can now find m_0, m_1 for which $\bar{H}(Y_0, m_0) = \bar{H}(Y_1, m_1)$ with probability $O(\min\{N^2, Q^2\} 2^{-\ell})$, where N is a bound on the number of possible messages in the system and Q is the total number of $\bar{H}$-oracle queries made by $\mathcal{F}$.

Therefore, if we envision a scenario in which the attacker may have access to the values Y before they are used by the signer $\hat{B}$ (as in the case in which pre-computed values of Y are leaked) then the security of the scheme depends on the quantity $O(\min\{N^2, Q^2\}2^{-\ell})$ rather than $2^{-\ell}$ as before. When the space of possible messages (in the case of HMQV this is the space of possible identities) is small or when $|q|$ is large enough (e.g., $|q| = 255$ as mandated in [36]) the above bound may be considered sufficient for dealing with this limited threat (see Remark 7.2). In other cases, however, one may want to strengthen HCR signatures against such leakage; we discuss several possible approaches. One, of course, is to increase ℓ. This however has a negative impact on the performance of HCR signatures (and their application to the HMQV protocol). A more efficient solution to the above problem is that instead of computing the exponent e as $\bar{H}(Y, m)$ one could compute $e = \bar{H}(Y, m, \nu)$ where ν is a nonce chosen by the signer at the time of signing and transmitted to the verifier as part of the signature.

Yet another solution, probably the most simple and economical, is to keep ℓ small, dispense of ν, and just include the value of the message m under the final hash. That is, instead of computing the HCR signature as $H(\sigma)$, one would compute it as $H(\sigma, m)$. In this case, the success probability of the above attack on an a-priori known Y would reduce to $\min\{N^2, Q^2\}2^{-(\ell+k)}$. If we consider a typical case (e.g., the use of HCR signatures in HMQV) where $\ell \geq 80$ and $k \geq 128$ then the level of security achieved by including m under the hash as above is perfectly satisfactory (like a birthday attack on a 208-bit hash function). In Section 7.4 we will discuss the possible effect of such a change to the security and specification of HMQV.

7.4 Application to HMQV: Resilience to the Leakage of DH Exponents

Here we show how the resilience of the HMQV protocol to the disclosure of ephemeral DH exponents follows from the proven security of HCR signatures. We first consider the case in which a party to a session generates the ephemeral DH value, say $Y = g^y$, during the run of the session and stores the exponent y in the session's state. In this case an attacker $\mathcal{M}$ can find the value y by breaking into the session's state (formally, via a state-reveal query). Later, we will consider the case in which the DH value Y is pre-computed ahead of the session in which it is used.

Lemma 29 *Under the GDH and KEA1 assumptions, protocol HMQV remains secure in the random oracle model even if session states are specified to store the secret exponent of the session's outgoing value. That is, learning the exponent of an ephemeral DH value used in a session via a state-reveal query does not help the attacker $\mathcal{M}$ in breaking other sessions (not even the session that uses the leaked ephemeral exponent).*

Proof (sketch): The lemma follows from Lemma 27 in a way similar to the proof of Lemma 10 (Section 5.2), namely, by showing how to construct a successful forger $\mathcal{F}$ in the game of Figure 5 given a successful HMQV attacker $\mathcal{M}$ which is allowed to learn ephemeral DH exponents via state-reveal queries. As usual, $\mathcal{F}$ will orchestrate a run of the HMQV protocol, in which the signing oracle $\hat{B}$ is used as one of the parties, and will run $\mathcal{M}$ against this simulated protocol. The difference with respect to previous such constructions of $\mathcal{F}$ is that we assume an attacker $\mathcal{M}$ that is allowed to learn, via state-reveal queries, the values of ephemeral exponents y (corresponding to DH values $Y = g^y$ exchanged in the protocol). Specifically, $\mathcal{F}$ needs to be able to provide these values for any session held at an uncorrupted party (except for the test session and its matching session). For uncorrupted parties other than $\hat{B}$, $\mathcal{F}$ itself chooses these exponents so it can provide them to $\mathcal{M}$. In the case of $\hat{B}$, $\mathcal{F}$ receives the exponents y directly from $\hat{B}$, as part of $\hat{B}$'s HCR signatures, and therefore it can provide these values to $\mathcal{M}$ if $\mathcal{M}$ queries them. In the case of state-reveal and session-key queries against sessions held at $\hat{B}$, $\mathcal{F}$ can also provide to $\mathcal{M}$ the value of the corresponding session keys since these are exactly the output of HCR signatures. As in the original proof in Section 5.2, the run of $\mathcal{M}$ by $\mathcal{F}$ is indistinguishable from a real run of $\mathcal{M}$ and therefore when $\mathcal{F}$ happens to guess the test session and $\mathcal{M}$ outputs a correct forgery for the test signature, then $\mathcal{F}$ can use this test signature as a forgery against $\hat{B}$. Thus, if $\mathcal{M}$ succeeds with non-negligible probability against HMQV, the constructed $\mathcal{F}$ succeeds also with non-negligible probability to forge HCR signatures. A final technical point is that since HMQV uses dual HCR signatures, in which the challenge X is replaced with XA^d for $d = \bar{H}(X, m)$, then in the above forgery game we have to replace challenges X with XA^d for $A = g^a$ and $d = \bar{H}(X, m)$. However, since the above proof works even in case that the attacker knows a then forging a signature with challenge XA^d is equivalent to forging it with challenge X. (Similarly to the modified computation of W (in this case, relative to Figure 6) as in the proof of Theorem 7). □

Remark 7.1 *(Security of sessions that use a compromised DH exponent.)* The proof of Lemma 29 shows that even if $\mathcal{M}$ learns the value y corresponding to a test session $(\hat{A}, \hat{B}, X, Y = g^y)$, $\mathcal{M}$ cannot distinguish the session key from random (this holds even if $\mathcal{M}$ knows $\hat{A}$'s private key a). In addition, it is possible to show that if both private keys of $\hat{A}$ and $\hat{B}$ are unknown to $\mathcal{M}$ and $\hat{A}$ completes a session $(\hat{A}, \hat{B}, X, Y)$ with values $X = g^x, Y = g^y$ chosen by $\hat{A}, \hat{B}$, respectively, then even if $\mathcal{M}$ knows x and y, the session key remains secure. The proof for the latter case consists of showing how to use an assumed successful attacker $\mathcal{M}$ in order to build a CDH solver that computes $CDH(A, B)$. We sketch this idea: Given inputs A, B, the solver $\mathcal{C}$ for $CDH(A, B)$ will run $\mathcal{M}$ against a setting of the HMQV protocol in which two of the parties (call them $\hat{A}, \hat{B}$) are provided with public keys A and B, respectively. The simulation of $\hat{A}$ and $\hat{B}$ (for which $\mathcal{C}$ does not know the private keys) uses the simulation of HCR signatures (helped by a DDH-oracle as before). If $\mathcal{M}$ succeeds in a test session $(\hat{A}, \hat{B}, X, Y)$ for which the exponents x, y are chosen by $\mathcal{C}$ on behalf of A, B (and possibly known to $\mathcal{M}$) then $\mathcal{M}$ must query the dual signature $g^{(x+da)(y+eb)}$ from the oracle H. From this value, and using the knowledge of x, y, d, e, the solver $\mathcal{C}$ derives $CDH(A, B) = g^{ab}$.

We proceed to analyze the case in which the exposure of the exponent y (or even g^y) happens before its use in a session. The security of the protocol in this case also follows from the unforgeability of HCR signatures. However, the setting that we need to consider is the one analyzed in Section 7.3, namely, in which the value g^y to be used in a session (or a signature) is known to the attacker in advance. As discussed there, in cases where ℓ is small and the space of messages (i.e., identities) is large, one may want to add an (unpredictable) nonce to the signature or to hash the signed message together with the hashing of σ. This results in two possible variants of HMQV:

1. When $\hat{A}$ sends $X = g^x$ to $\hat{B}$ she also sends a random nonce $\nu_{\hat{A}}$ (where $\nu_{\hat{A}}$ cannot be predicted by $\mathcal{M}$ before it is sent). Similarly, $\hat{B}$ will respond with $Y = g^y$ and its own unpredictable nonce $\nu_{\hat{B}}$. The exponents d and e are computed as $\bar{H}(X, \hat{B}, \nu_{\hat{A}})$ and $\bar{H}(Y, \hat{A}, \nu_{\hat{B}})$, respectively.

2. The protocol messages are unchanged; instead the parties' identities are added to the key derivation procedure; namely the session key is now computed as $K = H(\sigma, \hat{A}, \hat{B})$ instead of $K = H(\sigma)$ as before (the order of $\hat{A}$ and $\hat{B}$ under H can be determined arbitrarily but, of course, must be the same for both $\hat{A}$ and $\hat{B}$).

Both variants are simple and inexpensive. The second may be preferable as it saves the extra nonce; on the other hand, such a nonce is likely to be used anyway in actual implementations of the protocol for purposes of session identification (but note that the nonce in this case must be unpredictable). An advantage of variant 2 is that the hashing of identities in the key derivation function provides uniformity with the one-pass protocol of Section 9 (see Remark 9.1).

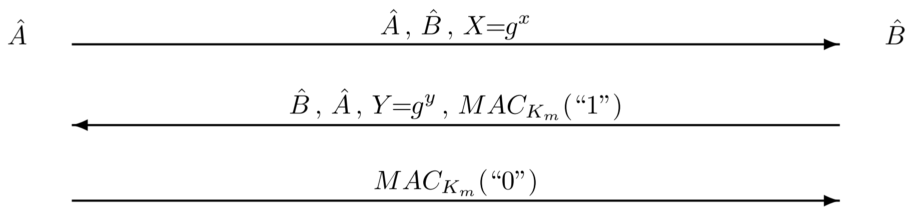
We thus have the following lemma.

Lemma 30 *Under the GDH and KEA1 assumptions, any one of the above variants of protocol HMQV remain secure in the random oracle model even if the attacker is allowed access to secret ephemeral DH exponents before or during the session that uses these exponents.*

Remark 7.2 *(The effect of birthday-type attacks on the original HMQV.)* While we do not recommend ignoring the effect of the birthday-type attacks described above (especially, that they can be avoided easily), it is illustrative to understand the effect of such an attack on the basic HMQV protocol. In order to take advantage of such an attack, $\mathcal{M}$ would need to know the value Y used by a party $\hat{B}$ in advance. Then it would need to find two identities in the system (recall that in HMQV the message m is always the identity of a party), say $\hat{P}$ and $\hat{P}'$, for which a collision $\bar{H}(Y_0, \hat{P}) = \bar{H}(Y, \hat{P}')$ can be found. Once these are found, the attacker can mount a UKS attack against $\hat{B}$ and $\hat{P}$ by forcing the generation of two unmatching sessions at these parties which output the same key but have different peers ($\hat{B}$ would end the session with peer $\hat{P}'$ and $\hat{P}$ with peer $\hat{B}$). In addition, for this attack to succeed, $\mathcal{M}$ would need to corrupt $\hat{P}'$ and register the same public key of $\hat{P}$ under the name of P' (this also assumes that no proof of possession is required by the CA). Finally, note that in general $\mathcal{M}$ may have some control about one of the identities to attack, but in order to control both identities it would require that each identity will have many different representations (e.g., the identity could include arbitrary random bits in its representation). In all, it can be argued that the practicality of such an attack is minimal. Yet, for the case in which ℓ is relatively small, say $\ell = 160$, then one may want to choose one of the above variants of HMQV.

8 The 3-message HMQV-C Protocol

The HMQV-C Protocol


$\hat{A}$
$\hat{A}$, $\hat{B}$, $X{=}g^x$
$\hat{B}$
$\hat{B}$, $\hat{A}$, $Y{=}g^y$, MAC_{K_m}(“1”)
MAC_{K_m}(“0”)

Key Computation (by $\hat{A}$ and $\hat{B}$):

Compute σ as in HMQV; compute $K_m = H(\sigma, 0)$ and $SK = H(\sigma, 1)$

Figure 7: $\hat{A}$ and $\hat{B}$, with public keys $A{=}g^a$ and $B{=}g^b$, resp., share session key SK.

In this section we study a 3-message variant of the HMQV protocol, which we refer to as HMQV-C (where C stands for “key Confirmation”), and which is depicted in Figure 7. The protocol enjoys all the security properties of HMQV (the same analysis of HMQV establishes these properties for HMQV-C) and essentially the same computational cost. It adds, however, a third message to the protocol and a slight increase in the length of protocol messages. In return, HMQV-C provides some properties lacking in the basic HMQV protocol. We discuss three such properties: key confirmation, PFS, and universal composability.

Key confirmation. The HMQV protocol provides a fundamental assurance to a party $\hat{A}$ that completes a session with peer $\hat{B}$ and session key K: if $\hat{B}$ is not corrupted then *only $\hat{B}$ may possibly* know K. What the protocol does not provide is any assurance to $\hat{A}$ that $\hat{B}$ completed the session or computed the session key; moreover, $\hat{B}$ might not have even been "alive" during the execution of the session. This is not just a drawback of HMQV but the same will be true for any 2-message public-key based protocol (assuming, as in the typical public key scenario, that no prior shared state was created at an earlier communication between $\hat{A}$ and $\hat{B}$). Furthermore, as pointed out by Shoup [41], the seemingly natural goal that *both* parties have assurance that the the peer completed the session before each starts using the key *cannot be achieved by any key-exchange protocol.* Indeed, the attacker can always prevent this mutual assurance by stopping the last protocol message from reaching its destination.

Yet, the weaker assurance to each of the parties that the peer was able to compute the key (but not necessarily that it output the key to the calling application) is achievable and referred to in the literature as the *key confirmation* property. While not crucial for the basic security of a key exchange (e.g., the lack of key confirmation is not a threat to the privacy or authenticity of communications protected with this key [11]), this property may provide a useful *"operational sanity check"* for some applications. In this case, protocol HMQV-C is better suited than HMQV since the added MAC values provide key confirmation. Moreover, the MAC validation confirms the active involvement of the identified peer to the session as well as the fact that this peer possesses a matching session (i.e., with same peers and same session key). Note that in order to achieve these properties, the MAC in HMQV-C does not need to be applied to any specific session information but simply to a single bit used to indicate the "direction" of the message and to prevent reflection. Also worth noting is that the protocol consisting of only the first two messages in HMQV-C already provides key confirmation to the initiator (which may add a useful feature to HMQV without increasing the number of protocol messages).

Note (denial of service): In many applications of key exchange, lack of key confirmation may lead to a form of "denial of service" (DoS) attack in which a party $\hat{A}$ starts using the key, say to send protected information to $\hat{B}$, while $\hat{B}$ is not able to process this information since it did not yet establish the key. As said, this situation cannot be avoided completely since mutual "session completion" confirmation is not achievable. Moreover, there are more serious forms of DoS attacks against protocols based on public key operations, in which a party is forced to spend significant computation cycles (and create session state) before discovering the invalidity of the peer. Some useful but limited-scope counter-measures to DoS attacks exist (cf. [19]) that can be applied to any key-exchange protocol (including HMQV) at the expense of added protocol messages.

Perfect Forward Secrecy (PFS). As shown in Section 6.2, no 2-message protocol, and in particular HMQV, can provide full perfect forward secrecy. Instead, the best one can hope for is the weak form of PFS provided by HMQV and proven in that section. The main advantage of HMQV-C is that it lifts this inherent limitation of HMQV and provides for full PFS (see formal definition in Section 2). The proof of this fact is similar to the proof of Lemma 23. In that case, in order to claim forward secrecy, we had to explicitly assume (as formally captured by the notion of "clean sessions") that the legitimate parties chose the DH values for the session. In the case of HMQV-C, however, there is no need for any such assumptions. Indeed, a completed session in HMQV-C already provides assurance that the DH values were chosen (and explicitly authenticated) by the session's peers. We thus obtain full perfect forward secrecy: if an uncorrupted party $\hat{A}$ establishes an HMQV-C session with uncorrupted peer $\hat{B}$, then the session key K remains secure even if the attacker corrupts $\hat{A}$ after K expired at $\hat{A}$, or it corrupts $\hat{B}$ after K expired at $\hat{B}$.

UC Security. In [13], Canetti and Krawczyk expand their model of key exchange from [11] (which we use as the basis for this work) to a more ambitious model targeted at ensuring the security of key-exchange protocols when run concurrently with other applications (as is the case in real-world environments). This model is known as the *Universal-Composability (UC) model of key exchange* (a part of the general UC framework of Canetti [10]). An important theorem from [13] shows that if a protocol is proven secure in the model of [11] and it also enjoys the so called "ACK property" then the protocol is also secure in the stronger UC model. Informally, the ACK property requires that when the first party to complete a session outputs its session key then the peer's state contains only information that can be "simulated" from the public information in the protocol and the session key. This can be shown to be the case for HMQV-C (see Claim 15 in [13]). Therefore, HMQV-C is secure in the UC model as well. (On the other hand, HMQV can be shown to be secure in the "relaxed UC" model from [13] but not in the regular UC model.)

We summarize the above properties of HMQV-C in the following theorem:

Theorem 31 *Under the CDH assumption, the HMQV-C protocol, with its hash function modeled as a random oracle, is a secure key-exchange protocol in the Canetti-Krawczyk model, which enjoys all the security properties from Theorem 18, and the additional properties of perfect forward secrecy (PFS) and key confirmation. The security of the protocol holds also in the Universal Composability model of key-exchange protocols [13].*

9 One-Pass HMQV

A one-pass key exchange protocol consists of a single message sent from a sender $\hat{A}$ to a recipient $\hat{B}$ from which both parties (using their private and public keys) derive a unique key that only $\hat{A}$ and $\hat{B}$ may possibly know (as long as both parties and the session are uncorrupted as defined below). In particular, $\hat{B}$ knows that the incoming message originated from $\hat{A}$ and was intended to $\hat{B}$. In other words, the requirements from the established key are the same as in a regular key exchange protocol except for the possibility that the message received by $\hat{B}$ is a replay of an older message from $\hat{A}$ (this replay is inevitable in a one-pass protocol, though it may be detectable by other means such as synchronized time or shared state). In addition, such a protocol cannot provide PFS since (by lack of a session-specific input from $\hat{B}$) the key should be computable with the sole knowledge of $\hat{B}$'s private key. Below we outline a formal definition of the security of one-pass protocols. The MQV papers [33, 29] suggested a one-pass variant of the protocol. Here we adopt the same ideas with respect to HMQV, and obtain a very efficient and provably secure one-pass key-exchange protocol.

One-pass HMQV. One-pass HMQV between parties $\hat{A}$ and $\hat{B}$ (with public keys $A = g^a, B = g^b$, respectively) consists of a single value $X = g^x$ transmitted from $\hat{A}$ to $\hat{B}$ (where $x \in_{\mathrm{R}} Z_q$ is chosen by $\hat{A}$). The session key K is computed by $\hat{A}$ as: (i) Let $(\hat{A}, \hat{B})$ denote a message that includes the two identities $\hat{A}$ and $\hat{B}$, and set d to be the result of $d = \bar{H}(X, (\hat{A}, \hat{B}))$. (ii) compute $\sigma_{\hat{A}} = \mathit{XSIG}_{\hat{A}}(X, (\hat{A}, \hat{B}), B) = B^{x+da}$ (XSIG is defined in Definition 2); (iii) set $K = H(\sigma_{\hat{A}})$ where H outputs a number of bits equal to the length of the required key. The same key K is computed by

$\hat{B}$ (after checking that $X \neq 0$) as $K = H((XA^d)^b)$.

Security model for one-pass protocols. The key-exchange security model of Canetti-Krawczyk outlined in Section 2 applies to the one-pass protocol case through the following changes. Sessions are named by their peers and the exchanged message, i.e. by the triple $(\hat{A}, \hat{B}, X)$ (for simplicity we use the same session notation for both $\hat{A}$ and $\hat{B}$). A session is called exposed if one of the following happens: (i) $\hat{A}$ or $\hat{B}$ are corrupted; (ii) a session-key query is issued against the session (in which case the attacker obtains the session key); (iii) a state-reveal query is issued against the session (either at $\hat{A}$ or $\hat{B}$). The test-session game is the same as in the general model (in particular, a test session can be chosen among any one of the unexposed sessions). The rest of the model and security definitions are unchanged.

Theorem 32 *Under the CDH assumption and in the random oracle model, One-Pass HMQV is a secure one-pass key-exchange protocol.*

The proof of this theorem combines two elements: (i) a proof based on XCR signatures (by $\hat{A}$) similar in spirit to the proof of the HMQV protocol from Section 5.2; and (ii) a proof of indistinguishability of the key K which involves arguments similar to those used in [1]. The latter aspect is orthogonal to most of the analysis work in this paper and its proof is related to other implicitly-authenticated Diffie-Hellman studied in [28]. The reason to include the identity of $\hat{A}$ under the signed message (i.e., under the hash that defines d) is to thwart a UKS-type attack as follows. The attacker $\mathcal{M}$ registers $\hat{A}$'s public key, A, as her own public key. Then, when $\hat{A}$ sends to $\hat{B}$ the message $\hat{A}, X$, the attacker $\mathcal{M}$ replaces it with $\mathcal{M}, X$. The result is that $\hat{B}$ believes the resultant key K is shared with $\mathcal{M}$ while $\hat{A}$ believes she shared (the same) K with $\hat{B}$.

We conclude by pointing out that the above one-pass protocol can be used as an *authenticated CCA encryption scheme*, that is, $\hat{A}$ can transmit a message m to $\hat{B}$ encrypted (against chosen-ciphertext attacks) as well as authenticated (by $\hat{A}$). For this, $\hat{A}$ would send a triple (X, c, t) where $X = g^x$, c is a ciphertext obtained as the symmetric CPA encryption of message m under a key K_1, and t a MAC value computed on c under key K_2. The keys K_1 and K_2 are derived from a key K computed from X as in the one-pass HMQV protocol. The whole cost of this procedure is 2 exponentiations for $\hat{A}$ (one is off-line) and 1.5 for $\hat{B}$. This is just 1/2 exponentiation more for $\hat{B}$ compared to the DHIES CCA encryption from [1] but in return it provides authentication from $\hat{A}$ (with DHIES this authentication would require a full additional signature from $\hat{A}$). This efficient authenticated CCA encryption is very attractive for ”store-and-forward” applications such as PGP, and significantly cheaper than the usual sign-and-encrypt paradigm. The only caveat here is that the identity $\hat{A}$ (and possibly its certificate) needs to be transmitted in the clear as it is needed for the decryption operation. Yet another property of the above protocol that is worth noting is that it can be used just as a verifier-specific signature of $\hat{A}$ on message m without necessarily adding the encryption part. This signature, however, is recipient specific and therefore does not provide non-repudiation; instead it provides deniability, a very valuable feature in many applications, e.g. PGP. (See also Remark 4.5.)

Remark 9.1 *(A note on standardization.)* Many of the standards that have adopted MQV, have also adopted the one-pass variant of it. For standards interested in adopting HMQV in its different forms (one, two and three messages), it could make sense to define the derivation of the key in the one-pass protocol similar to the derivation in the other variants of HMQV. Specifically, by substituting Y with B in the dual signatures that define the HMQV protocol (Section 5), we obtain the following values for the one-pass key: $\hat{A}$ and $\hat{B}$ compute $\sigma_{\hat{A}} = (BB^e)^{x+da}$ and $\sigma_{\hat{B}} = (XA^d)^{b+eb}$, respectively, and set the key K to the hash of these (equal) values. (Note that in this case the exponent e does not add any value to the protocol except for making it compatible with the other variants; it actually somewhat detracts from the protocol efficiency.) Yet, an additional discrepancy remains between the value of $d = \bar{H}(X, (\hat{A}, \hat{B}))$ in the one-pass version and $d = \bar{H}(X, \hat{B})$ in the 2-message version of HMQV. A way to provide compatibility between the three modes would be to have in all of them $d = \bar{H}(X, \hat{B}), e = \bar{H}(Y, \hat{A})$ (where $Y = B$ in the one-pass case), and add the identities $\hat{A}, \hat{B}$ to the session-key derivation function: namely, $K = H(\sigma, \hat{A}, \hat{B})$ (with the order of $\hat{A}$ and $\hat{B}$ defined using some fixed criterion). This replaces the need to add $\hat{A}$ in the computation of d. It also has the advantage, as discussed in Section 7.3 of strengthening HMQV in the case of leaked pre-computed DH values.

10 Concluding Remarks

The results in this paper show vulnerabilities of MQV to known-key and other attacks (in the case of the 3-message variant of MQV some of these vulnerabilities depend on the ability of the attacker to access ephemeral state information for incomplete sessions). The extent to which these weaknesses are exploitable in practice depends of course on the application, the computing and communication environment, threat model, etc. These results do *not* mean that applications already using MQV are necessarily insecure in their specific environments. At the same time, the identified weaknesses should not be dismissed as theoretical only, especially when considering that MQV has been (and is being) standardized as a key-exchange protocol for use in heterogeneous and unknown scenarios (including the highly sensitive applications such as those announced by the NSA [36]). This is particularly true when, as shown here, these weaknesses are not inherent to the problem being solved nor to the formal analytical setting.

Indeed, HMQV provides the same functionality with the same (or even better) performance of MQV while enjoying a full proof of security.[12] Two caveats regarding this proof are the use of the idealized random oracle methodology and the significant (though polynomially bounded) reduction cost. Other proven protocols (such as SKEME [26], ISO [22, 11] and SIGMA [27, 12]), while less efficient, enjoy less expensive reductions and do not directly require random oracles. We note, however, that dispensing of random oracles in these protocols requires the use of the more expensive (and seldom used in practice) signature and encryption schemes that do not rely on random oracles, and also requires the stronger DDH assumption. Certainly, coming up with a protocol that offers the many attractive security and performance properties of HMQV and does not rely on the random oracle model in its analysis is an important open question.

We end by stressing that in spite of the weaknesses demonstrated here, the MQV protocol contains some remarkable ideas without which the design of HMQV would have not been possible. Nor would this design have been possible without the rigorous examination of these ideas in a formal framework such as the one in [11]. The design of HMQV is a demonstration of the strength of "proof-driven designs" which guide us in choosing the necessary design elements of the protocol while dispensing of unnecessary "safety margins". As a result, one obtains solutions that are not only cryptographically sound but are also more efficient.

[12] We hope that standard bodies will take into account provability of protocols when selecting or revising protocols for standardization.

References

[1] M. Abdalla, M. Bellare and P. Rogaway, "The Oracle Diffie-Hellman Assumptions and an Analysis of DHIES", *CT-RSA 2001.* LNCS 2020, D. Naccache (Ed.), Springer-Verlag, 2001.

[2] American National Standard (ANSI) X9.42-2001, Public Key Cryptography for the Financial Services Industry: Agreement of Symmetric Keys Using Discrete Logarithm Cryptography.

[3] American National Standard (ANSI) X9.63: Public Key Cryptography for the Financial Services Industry: Key Agreement and Key Transport using Elliptic Curve Cryptography.

[4] M. Bellare and A. Palacio, "The Knowledge-of-Exponent Assumptions and 3-round Zero-Knowledge Protocols", *Crypto'04*, LNCS 3152,

[5] M. Bellare and P. Rogaway, "Random Oracles are Practical: A Paradigm for Designing Efficient Protocols", *First ACM Conference on Computer and Communications Security*, 1993.

[6] M. Bellare and P. Rogaway, "Entity authentication and key distribution", *Crypto'93*, LNCS 773, 1994, pp. 232-249.

[7] S. Blake-Wilson and A. Menezes, "Authenticated Diffie-Hellman Key Agreement Protocols", *Proceedings of SAC '99*, Lecture Notes in Computer Science, 1556 (1999),

[8] S. Blake-Wilson, D. Johnson and A. Menezes, "Key exchange protocols and their security analysis," *Sixth IMA International Conference on Cryptography and Coding*, 1997.

[9] C. Boyd and A. Mathuria, *Protocols for Authentication and Key Establishment*, Springer, 2003.

[10] R. Canetti, "Universally Composable Security: A New paradigm for Cryptographic Protocols", *42nd FOCS*, 2001.

[11] Canetti, R., and Krawczyk, H., "Analysis of Key-Exchange Protocols and Their Use for Building Secure Channels", Eurocrypt'2001, LNCS Vol. 2045.
Full version in: *Cryptology ePrint Archive* (`http://eprint.iacr.org/`), Report 2001/040.

[12] Canetti, R., and Krawczyk, H., "Security Analysis of IKE's Signature-based Key-Exchange Protocol", *Crypto 2002.* LNCS Vol. 2442.

[13] Canetti, R., and Krawczyk, H., "Universally Composable Notions of Key Exchange and Secure Channels", *Eurocrypt 02*, 2002. Full version available at `http://eprint.iacr.org/2002/059.`

[14] I. Damgård, "Towards Practical Public Key Systems Secure Against Chosen Ciphertext Attacks", Crypto'91, LNCS Vol. 576.

[15] W. Diffie and M. Hellman, "New Directions in Cryptography", IEEE Trans. Info. Theor. 22, 6 (Nov 1976), pp. 644–654.

[16] W. Diffie, P. van Oorschot and M. Wiener, "Authentication and authenticated key exchanges", *Designs, Codes and Cryptography*, 2, 1992, pp. 107–125.

[17] C. Dwork, M. Naor and A. Sahai, "Concurrent Zero-Knowledge", *proc. of* 30th *ACM Symposium on Theory of Computing (STOC)*, 1998, pp. 409–418

[18] Hada, S. and Tanaka, T., "On the Existence of 3-round Zero-Knowledge Protocols", Crypto'98, LNCS Vol. 1462.

[19] D. Harkins and D. Carrel, ed., "The Internet Key Exchange (IKE)", *RFC 2409,* Nov. 1998.

[20] IEEE 1363-2000: Standard Specifications for Public Key Cryptography.

[21] ISO/IEC IS 15946-3 "Information technology – Security techniques – Cryptographic techniques based on elliptic curves – Part 3: Key establishment", 2002.

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

[23] Ik Rae Jeong, Jonathan Katz, Dong Hoon Lee, "One-Round Protocols for Two-Party Authenticated Key Exchange", ACNS 2004: 220-232

[24] B. Kaliski, "An unknown key-share attack on the MQV key agreement protocol", *ACM Transactions on Information and System Security (TISSEC).* Vol. 4 No. 3, 2001, pp. 275–288.

[25] J. Katz, "Efficient and Non-Malleable Proofs of Plaintext Knowledge and Applications", *Advances in Cryptology – Proc. of EUROCRYPT '03, LNCS 2656*, Springer-Verlag, pp. 211–228, 2003.

[26] H. Krawczyk, "SKEME: A Versatile Secure Key Exchange Mechanism for Internet,", *1996 Internet Society Symposium on Network and Distributed System Security,* Feb. 1996.

[27] H. Krawczyk, "SIGMA: The 'SiGn-and-MAc' Approach to Authenticated Diffie-Hellman and Its Use in the IKE Protocols", *Crypto '03*, LNCS No. 2729, pp. 400–425, 2003.

[28] H. Krawczyk, "On the Security of Implicitly-Authenticated Diffie-Hellman Protocols", work in progress.

[29] 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.*

[30] T. Matsumoto, Y. Takashima, and H. Imai, "On seeking smart public-key distribution systems", *Trans. IECE of Japan*, 1986, E69(2), pp. 99-106.

[31] U. Maurer and S. Wolf, "Diffie-Hellman oracles", CRYPTO '96, LNCS, vol. 1109, 1996, pp. 268-282.

[32] A. Menezes, "Another Look at HMQV", In `http://eprint.iacr.org/2005/205`, June 27, 2005.

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

[34] A. Menezes, P. Van Oorschot and S. Vanstone, "Handbook of Applied Cryptography," CRC Press, 1996.

[35] NIST Special Publication 800-56 (DRAFT): Recommendation on Key Establishment Schemes. Draft 2, Jan. 2003.

[36] "NSAs Elliptic Curve Licensing Agreement", presentation by Mr. John Stasak (Cryptography Office, National Security Agency) to the IETF's Security Area Advisory Group, Nov 2004. http://www.machshav.com/˜smb/saag-11-2004/NSA-EC-License.pdf

[37] T. Okamoto and D. Pointcheval, "The Gap-Problems: A New Class of Problems for the Security of Cryptographic Schemes", *PKC 2001*, LNCS 1992, K. Kim (Ed.), Springer-Verlag, 2001, pp. 104–118.

[38] D. Pointcheval and J. Stern, "Security Arguments for Digital Signatures and Blind Signatures", *J.Cryptology* (2000) 13:361-396.

[39] M.O. Rabin, "Digitalized Signatures", Foundations of Secure Computing, DeMillo-Dobkins-Jones-Lipton, editors, 155-168, Academic Press, 1978

[40] V. Shoup, "Lower Bounds for Discrete Logarithms and Related Problems", Eurocrypt'97, LNCS 1233, pp. 256-266.

[41] V. Shoup, "On Formal Models for Secure Key Exchange", Theory of Cryptography Library, 1999. `http://philby.ucsd.edu/cryptolib/1999/99-12.html`.

A Kaliski's On-line UKS Attack Against MQV

For the benefit of the reader we present here Kaliski's UKS attack [24] demonstrating that the requirement of proof-of-possession from registrants of public keys is not a solution to this UKS weakness of the MQV protocol. Moreover, this attack demonstrates that replacing the exponents $d = \bar{X}, e = \bar{Y}$ used in the MQV protocol with $\bar{H}(X), \bar{H}(Y)$, respectively, for *any* function H (even a perfect random oracle) does not resolve the UKS attack.

1. $\hat{A}$ sends X to $\hat{B}$.
2. The attacker $\mathcal{M}$ intercepts X.
3. $\mathcal{M}$ registers with the CA a key $C = g^c$ where c is computed by (and then known to) $\mathcal{M}$ as:
   (a) Choose $u \in_{\mathrm{R}} Z_q$;
   (b) Set $d = H(X), Z = XA^d g^{-u}, h = H(Z)$, and $c = u/h$.
4. $\mathcal{M}$ sends Z to $\hat{B}$ as coming from $\mathcal{M}$ (with certified public key C).
5. $\hat{B}$ responds with Y to $\mathcal{M}$.
6. $\mathcal{M}$ relays Y to $\hat{A}$ as coming from $\hat{B}$.

Note that $ZC^h = XA^d$, and therefore the keys computed in sessions $(\hat{A}, \hat{B}, X, Y)$ and $(\hat{B}, \mathcal{M}, X, Y)$ (established by $\hat{A}$ and $\hat{B}$, respectively) are identical.

It is worth noting that the above specific attack can be prevented by making the values d, e, h depend on both exponents (e.g. $d = \bar{H}(X, Y)$), a UKS attack in this case is still possible (as pointed out in Section 3.2, even one that works against the 3-message MQV protocol with key confirmation). What is needed to solve the attack is to bind the outgoing DH value and the peer's identity in the definition of e and d, as done in HMQV.