Des. Codes Cryptogr. (2008) 46:329–342
DOI 10.1007/s10623-007-9159-1

# Obtaining a secure and efficient key agreement protocol from (H)MQV and NAXOS

**Berkant Ustaoglu**

Received: 27 March 2007 / Revised: 14 October 2007 / Accepted: 6 December 2007 /
Published online: 29 December 2007


**Abstract** LaMacchia, Lauter and Mityagin recently presented a strong security definition for authenticated key agreement strengthening the well-known Canetti-Krawczyk definition. They also described a protocol, called NAXOS, that enjoys a simple security proof in the new model. Compared to MQV and HMQV, NAXOS is less efficient and cannot be readily modified to obtain a one-pass protocol. On the other hand MQV does not have a security proof, and the HMQV security proof is extremely complicated. This paper proposes a new authenticated key agreement protocol, called CMQV ('Combined' MQV), which incorporates design principles from MQV, HMQV and NAXOS. The new protocol achieves the efficiency of HMQV and admits a natural one-pass variant. Moreover, we present a relatively simple and intuitive proof that CMQV is secure in the LaMacchia-Lauter-Mityagin model.

**Keywords** Key agreement protocols · MQV · Provable security

**AMS Classification** 94A60

## 1 Introduction

Researchers from IBM and Microsoft have recently proposed two-pass Diffie-Hellman authenticated key agreement protocols called HMQV [8], KEA+ [12] and NAXOS [11]. In these protocols the two communicating parties exchange static (long-term) and ephemeral (short-term) public keys, and thereafter combine them to obtain a session key. The papers [8,11,12] highlight certain security issues with previous related key agreement protocols and propose solutions to address those issues. The goal of this paper is to devise a new protocol that has the best of all worlds incorporated in its design.

Communicated by C. Boyd.

B. Ustaoglu (✉)
University of Waterloo, Waterloo, ON, Canada N2L 3G1
e-mail: bustaogl@math.uwaterloo.ca

**Table 1** Protocol comparison

| Protocol | Efficiency | Security | Assumptions |
|---|---|---|---|
| MQV | 2.5 | Unproven | ? |
| HMQV | 2.5 | CK, wPFS, KCI, LEP | KEA1, GDH, RO |
| KEA+ | 3 | CK, KCI | GDH, RO |
| NAXOS | 4 | eCK | GDH, RO |
| CMQV | 3 | eCK | GDH, RO |

*Security models and definitions.* Choo, Boyd and Hitchcock [7] compared the most commonly used security models for key agreement [3,4,6]. Their conclusion was that none of the models as defined provides a significant advantage over the rest of the models. Furthermore, these models fail to capture some desirable properties of key agreement. Most significantly, the adversary is not allowed to obtain certain secret information about the session that is being attacked. Krawczyk [8] addressed many of these concerns by providing a stronger version of the Canetti-Krawczyk model [6] that captures additional security properties. These desirable properties include resistance to key-compromise impersonation (KCI) attacks, weak perfect forward secrecy (wPFS), and resilience to the leakage of ephemeral private keys (LEP). More recently LaMacchia, Lauter and Mityagin [11] provided a single definition that *simultaneously* captures all these security properties. Their security model will henceforth be called the extended Canetti-Krawczyk (eCK) model.

*Protocols.* Currently only NAXOS is proven to be secure in the eCK model. NAXOS is less efficient in that it requires 4 exponentiations per party compared to 2.5 exponentiations for MQV and HMQV. In addition there is no natural modification of NAXOS to a one-pass protocol. Unlike MQV [13], the HMQV [8] protocol has a formal security proof.[1] However the proof is extremely long and complicated, and some significant (but fixable) flaws [15,16] have been discovered. The security proof for KEA+ [12] is in a model that is weaker than eCK; for example the adversary is not allowed to obtain the static private keys of both communicating parties. Table 1 compares MQV, HMQV, KEA+ and NAXOS in terms of efficiency (number of exponentiations per party), security and underlying assumptions. (See Sect. 3.3 for a more detailed analysis of the efficiency of CMQV.) As usual CK stands for Canetti-Krawczyk [6], GDH refers to the Gap Diffie-Hellman assumption [18], RO is short for the random oracle model, and KEA1 is the knowledge of exponent assumption [2].

*Goals.* This paper presents the two-pass CMQV protocol that achieves the following objectives: (i) intuitive design principles; (ii) efficiency of MQV and HMQV; (iii) relatively straightforward security proof with minimal assumptions in the eCK model; and (iv) a natural one-pass variant. The security proof was inspired by the HMQV argument [8], however NAXOS' idea of hashing ephemeral private keys with static private keys is essential to show security in the eCK model. Moreover, unlike the HMQV proof, the CMQV security proof does not need the KEA1 assumption in order to demonstrate resilience to the leakage of ephemeral private keys. On the negative side, the security of CMQV is *not* tight. As in the case of HMQV, the reduction uses the Forking Lemma of Pointcheval and Stern [19], which results in a highly non-tight reduction.

*Organization.* Section 2 outlines the extended Canetti-Krawczyk security model and formalizes the security definition. The CMQV protocol is described in Sect. 3, and the complete security proof is provided in Sect. 4. Finally, the one-pass variant of CMQV is presented in Sect. 5.

[1] The security proof for MQV presented in [10] is in a very restricted security model.

*Notation.* Let $q$ be a prime, and let $\mathbb{Z}_q$ denote the integers modulo $q$. We denote by $\mathcal{G} = \langle G \rangle$ a multiplicatively-written cyclic group of order $q$ generated by $G$, and by $\mathcal{G}^*$ the set of non-identity elements in $\mathcal{G}$. For group elements $A, B, \ldots$ the corresponding lowercase letters will denote the discrete logarithms in base $G$; for example $a = \log_G A$, where $a \in \mathbb{Z}_q$. Key agreement protocols take place between two parties, from among a set of $n$ parties, denoted by $\hat{A}$, $\hat{B}$ and so on. Party $\hat{A}$'s static public key is $A \in \mathcal{G}$ and its corresponding static private key is $a = \log_G A$. In general, lower case letters represent secret information, whereas upper case letters are publicly known values. Finally, the symbol "$\in_R$" means "selected uniformly at random".

## 2 Extended Canetti-Krawzcyk security model

In this section we outline the eCK model; for further details the reader is referred to [6,11]. In the eCK model there are $n$ parties each modeled by a probabilistic Turing machine. Each party has a static public-private key pair together with a certificate that binds the public key to that party. We do not assume that the certifying authority (CA) requires parties to prove possession of their static private keys, but we insist that the CA verifies that the static public key of a party belongs to $\mathcal{G}^*$. For simplicity, we will only describe the model for two-pass Diffie-Hellman protocols that exchange ephemeral and static public keys—this is without loss of generality as all the protocols in Table 1 are of this kind. More precisely, two parties $\hat{A}$, $\hat{B}$ exchange static public keys $A, B \in \mathcal{G}^*$ and ephemeral public keys $X, Y \in \mathcal{G}^*$; the session key is obtained by combining $A, B, X, Y$ and possibly the identities $\hat{A}, \hat{B}$.

*Sessions.* A party $\hat{A}$ can be activated to execute an instance of the protocol called a *session*. Activation is made via an incoming message that has one of the following forms: (i) $(\hat{A}, \hat{B})$ or (ii) $(\hat{A}, \hat{B}, Y)$. If $\hat{A}$ was activated with $(\hat{A}, \hat{B})$ then $\hat{A}$ is the session *initiator*, otherwise the session *responder*. If $\hat{A}$ is the responder of a session then $\hat{A}$ prepares an ephemeral public key $X$ and creates a separate session *state* where all session-specific ephemeral information is stored. The session is identified via a session *identifier* $(\hat{A}, \hat{B}, X, Y)$. If $\hat{A}$ is the initiator of a session, then $\hat{A}$ prepares an ephemeral public key $X$ and creates a session state as in the responder case. At the onset of the protocol the initiator does not know the incoming ephemeral public key. However the session can be uniquely[2] identified with $(\hat{A}, \hat{B}, X, \times)$, and hence this string can be used as the (temporary and incomplete) session identifier. When $\hat{A}$ receives the corresponding ephemeral public key $Y$, the session identifier is updated to $(\hat{A}, \hat{B}, X, Y)$. A session $(\hat{B}, \hat{A}, Y, X)$ (if it exists) is said to be *matching* to the session $(\hat{A}, \hat{B}, X, \times)$. On the other hand, the session matching to $(\hat{A}, \hat{B}, X, Y)$ can be any session identified by $(\hat{B}, \hat{A}, Y, \times)$ or $(\hat{B}, \hat{A}, Y, X)$. Since it is not possible (except with negligible probability) to simultaneously have two different sessions with identifiers $(\hat{B}, \hat{A}, Y, \times)$ and $(\hat{B}, \hat{A}, Y, X)$, a session $(\hat{A}, \hat{B}, X, Y)$ can have at most one matching session. For a session $(\hat{A}, \hat{B}, *, *)$, we call $\hat{A}$ the *owner* of the session, and $\hat{B}$ the *peer* of the session.

*Adversary.* The adversary $\mathcal{M}$ is modeled as a probabilistic Turing machine and controls *all* communications. Parties submit outgoing messages to the adversary, who makes decisions about their delivery. The adversary presents parties with incoming messages via *Send*(message), thereby controlling the activation of sessions. The adversary does not have immediate access to a party's private information, however in order to capture possible leakage of private information the adversary is allowed the following queries:

[2] Since ephemeral keys are selected at random on a per-session basis, the probability that an ephemeral public key $X$ is chosen twice by $\hat{A}$ is negligible.

- *EphemeralKeyReveal*(*sid*)—The adversary obtains the ephemeral private key held by the session *sid*.
- *SessionKeyReveal*(*sid*)—The adversary obtains the session key for a session *sid*, provided that the session holds a session key.
- *StaticKeyReveal*(party)—The adversary learns the static private key of the party.
- *EstablishParty*(party)—This query allows the adversary to register a static public key on behalf of a party. In this way the adversary totally controls that party. Parties against whom the adversary did not issue this query are called *honest*.

*Adversary goal.* The aim of the adversary $\mathcal{M}$ is to distinguish a session key from a random key. Formally, the adversary is allowed to make one special query *Test*(*sid*). The adversary is then given with equal probability either the session key held by *sid* or a random key. The adversary wins the game if he guesses correctly whether the key is random or not. To define secure protocols we need the following.

**Definition 2.1** (*fresh session*) Let *sid* be the session identifier of a completed session, owned by an honest party $\hat{A}$ with peer $\hat{B}$, who is also honest. Let $sid^*$ be the session identifier of the matching session of *sid*, if it exists. Define *sid* to be *fresh* if none of the following conditions hold:

(i) $\mathcal{M}$ issues a *SessionKeyReveal*(*sid*) query or a *SessionKeyReveal*($sid^*$) query (if $sid^*$ exists);
(ii) $sid^*$ exists and $\mathcal{M}$ makes either of the following queries:
  – both *StaticKeyReveal*($\hat{A}$) and *EphemeralKeyReveal*(*sid*), or
  – both *StaticKeyReveal*($\hat{B}$) and *EphemeralKeyReveal*($sid^*$);
(iii) $sid^*$ does not exist and $\mathcal{M}$ makes either of the following queries:
  – both *StaticKeyReveal*($\hat{A}$) and *EphemeralKeyReveal*(*sid*), or
  – *StaticKeyReveal*($\hat{B}$).

We are now ready to present the eCK security notion.

**Definition 2.2** (*eCK security*) A key agreement protocol is *secure* if the following conditions hold:

(i) If two honest parties complete matching sessions then, except with negligible probability, they both compute the same session key (or both output indication of protocol failure).
(ii) No polynomially bounded adversary $\mathcal{M}$ can distinguish the session key of a fresh session from a randomly chosen session key, with probability greater than 1/2 plus a negligible fraction.

The adversary $\mathcal{M}$ is allowed to continue interacting with the parties even after issuing the *Test* query. However, the test session must remain fresh through the experiment.

As mentioned at the end of Sect. 1, this security definition is very strong in the sense that it simultaneously captures most of the desirable security properties for authenticated key agreement that have been identified in the literature including resistance to key-compromise impersonation attacks, weak perfect forward secrecy, and resilience to the leakage of ephemeral private keys. Unlike in the CK model [6], the adversary in the eCK model is not equipped with a *SessionStateReveal* query which enables it to learn the entire session state of a particular session. This does not represent a deficiency in the eCK model since protocols such as HMQV [8] proven secure in the CK model typically specify that the

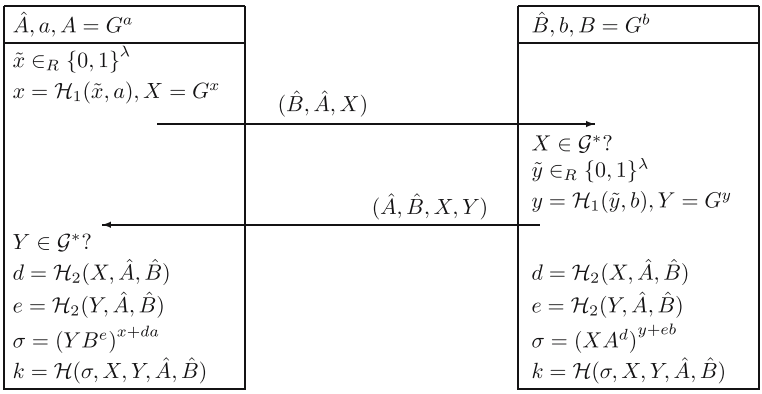


**Fig. 1** Fig. Two-pass CMQV

ephemeral private key is the only private information stored in the session state in which case the *EphemeralKeyReveal* query is functionally equivalent to the *SessionStateReveal* query. In general, by specifying that the session specific private information (the session state) is part of the ephemeral private key, the *SessionStateReveal* and *EphemeralKeyReveal* queries can be made functionally equivalent.

## 3 Two-pass CMQV

Two-pass CMQV is a Diffie-Hellman authenticated key agreement protocol that aims to establish a secure session key between two parties; see Fig. 1 for an informal description. In addition to the notation adopted at the end of Sect. 1, let $\mathcal{H}_1 : \{0,1\}^\lambda \times \mathbb{Z}_q^* \to \mathbb{Z}_q^*$, $\mathcal{H}_2 : \{0,1\}^* \to \mathbb{Z}_q$, and $\mathcal{H} : \{0,1\}^* \to \{0,1\}^\lambda$ be hash functions modeled as random oracles.

### 3.1 Protocol description

We assume for the remainder of the paper that a party never executes the protocol with itself. The two-pass CMQV protocol is formally given in the following.

**Definition 3.1** (*two-pass CMQV protocol*) Party $\hat{A}$'s static private key is $a \in_R [1, q-1]$ and $\hat{A}$'s static public key is $A = G^a$. Similarly, $\hat{B}$'s static key pair is $(b, B)$. To establish a session key with $\hat{B}$, party $\hat{A}$ performs the following steps.

1. Select an ephemeral private key $\tilde{x} \in_R \{0,1\}^\lambda$.
2. Compute $x = \mathcal{H}_1(\tilde{x}, a)$ and the ephemeral public key $X = G^x$.
3. Destroy $x$.
4. Send $(\hat{B}, \hat{A}, X)$ to $\hat{B}$.

Upon receiving $(\hat{B}, \hat{A}, X)$, party $\hat{B}$ verifies that $X \in \mathcal{G}^*$. If so, $\hat{B}$ performs the following steps.

1. Select an ephemeral private key $\tilde{y} \in_R \{0,1\}^\lambda$.
2. Compute $y = \mathcal{H}_1(\tilde{y}, b)$ and the ephemeral public key $Y = G^y$.
3. Destroy $y$.
4. Send $(\hat{A}, \hat{B}, X, Y)$ to $\hat{A}$.

Upon receiving $(\hat{A}, \hat{B}, X, Y)$, party $\hat{A}$ checks if he owns a session with identifier $(\hat{A}, \hat{B}, X, \times)$. If so, $\hat{A}$ verifies that $Y \in \mathcal{G}^*$.

To compute the session key both parties compute

$$d = \mathcal{H}_2(X, \hat{A}, \hat{B}), \quad e = \mathcal{H}_2(Y, \hat{A}, \hat{B}).$$

In addition $\hat{A}$ computes

$$\sigma = (YB^e)^{\mathcal{H}_1(\tilde{x},a)+da}$$

and $\hat{B}$ computes

$$\sigma = (XA^d)^{\mathcal{H}_1(\tilde{y},b)+eb}.$$

If all checks are satisfied the session key is

$$k = \mathcal{H}(\sigma, X, Y, \hat{A}, \hat{B}).$$

It is straightforward to verify that both parties compute the same shared secret $\sigma$, and therefore also the same session key.

### 3.2 Design rationale

In this section we explain the underlying principles behind the design of the CMQV protocol.

*Public-key validation.* Public-key validation (i.e. checking that static and ephemeral public keys belong to $\mathcal{G}^*$) prevents potential invalid-curve [1] and small subgroup attacks [14] (see also [16]). In other words, with validation a party obtains some assurance that computations involving its static private key do not reveal any information about the key itself, as long as the underlying group is cryptographically strong.

*Hashing ephemeral and static private keys.* A careful reader observes that in Definition 3.1 the value $x = \mathcal{H}_1(\tilde{x}, a)$ is *never* stored. Whenever $\mathcal{H}_1(\tilde{x}, a)$ is needed, it is computed. This implies that the session state does not store $x$. The idea is that without knowing *both* the ephemeral private key $\tilde{x}$ *and* the static private key $a$, no entity is able to compute the discrete logarithm $x$ of an ephemeral public key $X$. This elegant idea, first described in [11], allows the protocol to attain resistance to ephemeral private key leakage without resorting to non-trivial assumptions like KEA1 (as needed for HMQV [8]).

*Rationale for e and d.* Given a Computational Diffie-Hellman challenge with inputs $U, V \in_R \mathcal{G}$, knowledge of either of the discrete logarithms of $U$ or $V$ is enough to solve the CDH instance. If an adversary $\mathcal{M}$, given a static public key $B$, is able to find a group element $Y$ such that $\mathcal{M}$ knows the discrete logarithm of $T = YB^e$, then it is easy to see that $\mathcal{M}$ can impersonate $\hat{B}$ to other parties (since $\mathcal{M}$ can compute the shared secret $\sigma = (XA^d)^t$ where $t = \log_G T$, thereby impersonating $\hat{B}$ to $\hat{A}$). Defining $e$ to depend on $Y$ ensures that the adversary is not able to compute the discrete logarithm of $YB^e$. Moreover, including the identity of the intended peer in the derivation of $e$ prevents the adversary from potentially benefiting from the replay of $Y$ to two distinct parties $\hat{A}$ and $\hat{C}$. One may argue that the inclusion of $\hat{B}$'s identity in the derivation of $e$ is not needed since $\sigma$ in any case depends on $\hat{B}$'s static public key $B$. However, since the CA does not require parties to prove possession of their static private keys, $\mathcal{M}$ may establish a new party with static public key $B$. Hence $\hat{B}$ is included in the derivation of $e$.

We note that a very similar definition of $e$ was used in HMQV [8]. For both HMQV and CMQV, this definition of the exponents is crucial for the security proof, but in both cases

**Table 2** Efficiency comparison in terms of group exponentiations

| | DSA groups | Elliptic curves of prime order | Elliptic curves of nearly prime order |
|---|---|---|---|
| CMQV | 3.25 (4) | 2.25 (3) | 2.25 (3) |
| MQV | 3.25 (3.5) | 2.25 (2.5) | 2.25 (2.5) |
| HMQV-P1363 | 2.5/3.5 | 2.25 | 2.25 |

the reduction is non-tight. It is worth investigating if the requirements on $e$ and $d$ can be modified to attain a tight security reduction.

*Session key derivation.* The session key is $k = \mathcal{H}(\sigma, X, Y, \hat{A}, \hat{B})$. The secrecy of $\sigma$ guarantees that only the intended parties can possibly compute $k$. Including identities in the key derivation is a generic way to prevent unknown-key share attacks (see [5]). Furthermore, inclusion of $X$ and $Y$ in the key derivation allows for a simple argument that non-matching sessions have different session keys.

### 3.3 Efficiency of CMQV

The efficiency comparison in Table 1 is simplified; in particular, it does not take into account validation and various speedups that may be applicable. Consider the following groups of practical interest: (i) DSA-type groups (order-$q$ subgroups of the multiplicative group of prime fields $\mathbb{F}_p$); and (ii) elliptic curves of prime order $q$ or nearly prime order $hq$. Validation for DSA-type groups requires a full exponentiation; in contrast validating points on elliptic curves of prime order is essentially free. For nearly prime order curves, rather than verifying that the order of a public key is $q$, parties could use the corresponding public keys multiplied by the cofactor $h$. If the two public keys $Y$ and $B$ are validated, then computing $(YB^e)^{x+da}$ is equivalent to computing $Y^{s_1}B^{s_2}$, where $s_1 = x + da \bmod q$ and $s_2 = e(x + da) \bmod q$. Therefore, CMQV computations can be speedup using Shamir's trick (Algorithm 14.88 in [17]), reducing the cost by 0.75 exponentiations on average.

Table 2 compares CMQV with HMQV as described in [9], accounting for the validation and Shamir's speedup. The numbers in parentheses for MQV and CMQV represent the naive count of group exponentiations without accounting for possible improvements in the computations. The numbers for HMQV correspond to the two versions of HMQV as described in [9]. For HMQV, the difference is significant only in DSA-type groups as the more efficient version avoids full validation. However, the security proof in the case where validation is not required assumes that no ephemeral private keys are leaked to the adversary.

## 4 Security of CMQV

This section presents a formal security argument for two-pass CMQV. The *GDH assumption* in $\mathcal{G}$ is that the CDH problem in $\mathcal{G}$ cannot be solved in polynomial time with non-negligible success probability even when a DDH oracle for $\mathcal{G}$ is available.

**Theorem 4.1** *If $\mathcal{H}_1$,$\mathcal{H}_2$ and $\mathcal{H}$ are random oracles, and $\mathcal{G}$ is a group where the GDH assumption holds, then CMQV is eCK secure.*

*Proof* Let $\lambda$ denote the security parameter, whence $q = |\mathcal{G}| = \Theta(2^\lambda)$. Let $\mathcal{M}$ be a polynomially (in $\lambda$) bounded CMQV adversary. We say that $\mathcal{M}$ succeeds with non-negligible

probability if $\mathcal{M}$ wins the distinguishing game described in Sect. 2 with probability $\frac{1}{2} + p(\lambda)$, where $p(\lambda)$ is non-negligible. Assume that $\mathcal{M}$ succeeds with non-negligible probability in an environment that involves at most $n(\lambda)$ parties, $\mathcal{M}$ activates at most $s(\lambda)$ sessions, and makes at most $h_1(\lambda)$, $h_2(\lambda)$ and $h(\lambda)$ queries to oracles $\mathcal{H}_1$,$\mathcal{H}_2$ and $\mathcal{H}$ respectively. A guess for the answer to the *Test* query succeeds with probability $\frac{1}{2}$. Since $\mathcal{H}$ is a random oracle, $\mathcal{M}$ can only distinguish a session key from a random string with probability significantly greater than $\frac{1}{2}$ in one of the following ways:

A1. Guessing attack: $\mathcal{M}$ correctly guesses the session key.
A2. Key replication: $\mathcal{M}$ forces two distinct non-matching sessions to have the same session key. In that case $\mathcal{M}$ can select one of the sessions as the test session and query the key of the other session.
A3. Forging attack: $\mathcal{M}$ computes $\sigma$ and queries $\mathcal{H}$ with $(\sigma, X, Y, \hat{A}, \hat{B})$.

The probability of guessing the output of $\mathcal{H}$ is $\mathcal{O}(1/2^{\lambda})$, which is negligible; thus event A1 can be ruled out.

The input to the key derivation includes all information contained in the session identifier. Since two non-matching sessions cannot have the same communicating parties and ephemeral public keys (except with negligible probability), key replication is equivalent to finding a collision for $\mathcal{H}$. Therefore event A2 occurs with probability $\mathcal{O}(s(\lambda)^2/2^{\lambda})$, which is negligible.

It remains to consider event A3—forging attacks. The rest of this section is devoted to the analysis of this event. Following the standard approach we will show how to construct a GDH solver, that uses an adversary $\mathcal{M}$ that succeeds with non-negligible probability in a forging attack. The solver $\mathcal{S}$ is given a CDH challenge $(U, V)$, where $U, V \in_R \mathcal{G}$, and its task is to compute $\mathrm{CDH}(U, V) = U^v = V^u$. The solver is also given a DDH oracle for $\mathcal{G}$, which on input $(R_1, R_2, R_3)$, where $R_1, R_2, R_3 \in \mathcal{G}$, returns 1 if $R_3 = \mathrm{CDH}(R_1, R_2)$ and 0 otherwise.

Let $M$ denote the event that $\mathcal{M}$ is successful, $H$ denote the event that $\mathcal{M}$ queries $\mathcal{H}$ with $(\sigma, X, Y, \hat{A}, \hat{B})$, and $\overline{H}$ denote the complement of event $H$. By definition event A3 is equal to $(M \wedge H)$. Since $\mathcal{H}$ is a random oracle and events A1 and A2 were ruled out, we have that $\Pr(M \mid \overline{H}) = \frac{1}{2}$, where negligible terms were ignored. Hence

$$\Pr(M) = \Pr(M \wedge H) + \Pr(M \mid \overline{H})\Pr(\overline{H}) \leq \Pr(M \wedge H) + \frac{1}{2},$$

and therefore $\Pr(\mathrm{A3}) \geq p(\lambda)$. Consider the following complementary events:

E1. There exists an honest party $\hat{B}$ such that $\mathcal{M}$, during its execution, queries $\mathcal{H}_1$ with $(*, b)$, before issuing a *StaticKeyReveal*$(\hat{B})$ query. (Note that $\mathcal{M}$ does not necessarily make a *StaticKeyReveal*$(\hat{B})$ query.)
E2. During its execution, for every honest party $\hat{B}$ for which $\mathcal{M}$ queries $\mathcal{H}_1$ with $(*, b)$, $\mathcal{M}$ issued *StaticKeyReveal*$(\hat{B})$ before the first $(*, b)$ query to $\mathcal{H}_1$.

If $\mathcal{M}$ succeeds in a forging attack with non-negligible probability then either $(\mathrm{A3} \wedge \mathrm{E1})$ or $(\mathrm{A3} \wedge \mathrm{E2})$ occurs with non-negligible probability. These two events will be considered separately.

### 4.1 Event E1

We use $\mathcal{M}$ to construct an algorithm $\mathcal{S}$ that succeeds with non-negligible probability provided that the event $(\mathrm{A3} \wedge \mathrm{E1})$ occurs. In this case $\mathcal{S}$ prepares $n(\lambda)$ parties. One party, which we call

$\hat{B}$, is selected at random and assigned static public key $V$. (Recall that $V$ is one of the group elements from the CDH challenge.) The remaining $n(\lambda)$—1 parties are assigned random static key pairs. The adversary $\mathcal{M}$ is initiated on this set of parties and $\mathcal{S}$ awaits the actions of $\mathcal{M}$. When $\mathcal{M}$ activates a party whose static private key $\mathcal{S}$ possesses, then $\mathcal{S}$ follows the protocol description. We next discuss the actions of $\mathcal{S}$ when $\mathcal{M}$ queries the random oracles $\mathcal{H}_1$, $\mathcal{H}_2$ or $\mathcal{H}$, or when $\mathcal{M}$ activates sessions owned by $\hat{B}$.

1. *Send*$(\hat{B}, \hat{C})$: $\mathcal{S}$ selects $\tilde{y} \in_R \{0,1\}^\lambda$ and $y \in_R \mathbb{Z}_q^*$, and computes $Y = G^y$. Then $\mathcal{S}$ creates a new session with identifier $(\hat{B}, \hat{C}, Y, \times)$ and presents $\mathcal{M}$ with the outgoing message $(\hat{C}, \hat{B}, Y)$.
2. *Send*$(\hat{B}, \hat{C}, X)$: $\mathcal{S}$ selects $\tilde{y} \in_R \{0,1\}^\lambda$ and $y \in_R \mathbb{Z}_q^*$, computes $Y = G^y$, and creates a new session with identifier $(\hat{B}, \hat{C}, Y, X)$. Furthermore, if $\hat{C}$ is an honest party, then $\mathcal{S}$ checks if $\hat{C}$ owns a matching session; in this case $\mathcal{S}$ computes the shared secret $\sigma$ using the information in the matching session and assigns a random session key $k = \mathcal{H}(\sigma, X, Y, \hat{C}, \hat{B})$. If on the other hand, $\hat{C}$ is not honest, or if $\hat{C}$ does not own a matching session, then $\mathcal{S}$ assigns a random session key $k = \mathcal{H}(\times, X, Y, \hat{C}, \hat{B})$. Finally, $\mathcal{S}$ presents $\mathcal{M}$ with the outgoing message $(\hat{C}, \hat{B}, X, Y)$.
3. *Send*$(\hat{B}, \hat{C}, Y, X)$: $\mathcal{S}$ checks if $\hat{B}$ owns a session with identifier $(\hat{B}, \hat{C}, Y, \times)$; if not the session is aborted. Otherwise, $\mathcal{S}$ updates the identifier to $(\hat{B}, \hat{C}, Y, X)$. Next, $\mathcal{S}$ checks if $\mathcal{H}$ was queried with $(\sigma, Y, X, \hat{B}, \hat{C})$, where $\mathrm{DDH}(YB^e, XC^d, \sigma) = 1$, in which case the answer to that query is selected as the session key $k$. (Such a query was made if the session has a matching session owned by an honest party whose static private key $\mathcal{S}$ possesses, or by an adversary controlled party.) If no such query was made then $\mathcal{S}$ assigns a random session key $k = \mathcal{H}(\times, Y, X, \hat{B}, \hat{C})$.
4. $\mathcal{H}_1(s, c)$: $\mathcal{S}$ checks if $G^c = V$; if the equation holds then $\mathcal{S}$ stops $\mathcal{M}$ and computes $U^c = \mathrm{CDH}(U, V)$. In all other cases queries are answered in the usual way (by returning a random value from $\mathbb{Z}_q^*$ for new queries, and replaying answers if the queries were made before).
5. *EphemeralKeyReveal*(*sid*): $\mathcal{S}$ submits the value $\tilde{y}$ selected for *sid*.
6. *SessionKeyReveal*(*sid*): $\mathcal{S}$ submits the session key held by *sid*.
7. *StaticKeyReveal*$(\hat{B})$ or *EstablishParty*$(\hat{B})$: $\mathcal{S}$ aborts.
8. $\mathcal{H}_2$ queries: $\mathcal{S}$ simulates a random oracle in the usual way.
9. $\mathcal{H}$ queries: $\mathcal{S}$ simulates a random oracle in the usual way except for queries of the form $\mathcal{H}(\sigma, X, Y, \hat{C}, \hat{B})$ and $\mathcal{H}(\sigma, Y, X, \hat{B}, \hat{C})$. The problem with answering these queries is that it is possible that $\mathcal{S}$ had to assign a value to an $\mathcal{H}$ query for which $\mathcal{S}$ did not know all of the input (cf. items 2 and 3 above). In particular, there may exist queries recorded as $\mathcal{H}(\times, X, Y, \hat{C}, \hat{B})$ or $\mathcal{H}(\times, Y, X, \hat{B}, \hat{C})$ for which the first entry is not known to $\mathcal{S}$. For these queries $\mathcal{S}$ verifies that $\mathrm{DDH}(XC^d, YB^e, \sigma) = 1$. If so, $\mathcal{S}$ returns the previously selected string; otherwise $\mathcal{S}$ returns a random value.
10. *Test*(*sid*): $\mathcal{S}$ selects $\gamma \in_R \{0,1\}$. If $\gamma = 0$ then $\mathcal{S}$ returns a randomly chosen key; otherwise $\mathcal{S}$ returns the session key held by *sid*.

*Analysis.* It is easy to see that $\mathcal{S}$'s simulation of $\mathcal{M}$'s environment is perfect except with negligible probability. With probability at least $1/n(\lambda)$, $\mathcal{S}$ assigns the public key $V$ to $\hat{B}$, where $\hat{B}$ is an honest party for whom $\mathcal{M}$ will query $\mathcal{H}_1(*, b)$ without first issuing a *StaticKeyReveal*$(\hat{B})$ query. In this case the abortion in item 7 does not occur. Suppose event (A3 $\wedge$ E1) occurs, then at a certain stage $\mathcal{M}$ queries $\mathcal{H}_1$ with $(*, v)$, in which case $\mathcal{S}$ is successful as described in item 4. The probability that $\mathcal{S}$ is successful is

$$\Pr(\mathcal{S}) \geq \frac{1}{n(\lambda)} p_1(\lambda), \tag{1}$$

where $p_1(\lambda)$ is the probability of event (A3 $\wedge$ E1) and negligible terms are ignored. The simulation requires $\mathcal{S}$ to perform group exponentiations, access the DDH oracle, and simulate random oracles. Since $q = \Theta(2^\lambda)$, a group exponentiation takes time $\mathcal{T}_\mathcal{G} = \mathcal{O}(\lambda)$ group multiplications, whereas a DDH oracle call takes time $\mathcal{T}_{\text{DDH}} = \mathcal{O}(1)$. Responding to an $\mathcal{H}$ query takes time $\mathcal{T}_\mathcal{H} = \mathcal{O}(1)$; and similarly for $\mathcal{H}_1$ and $\mathcal{H}_2$ queries. Therefore, the running time of $\mathcal{S}$ is bounded by

$$\mathcal{T}_\mathcal{S} \leq \big(\mathcal{T}_\mathcal{G} + (\mathcal{T}_{\text{DDH}} + 2\mathcal{T}_\mathcal{G} + \mathcal{T}_\mathcal{H}) + (\mathcal{T}_\mathcal{G} + \mathcal{T}_{\mathcal{H}_1}) + \mathcal{T}_{\mathcal{H}_2}\big)\mathcal{T}_\mathcal{M}, \tag{2}$$

where $\mathcal{T}_\mathcal{M}$ is $\mathcal{M}$'s running time. Since all term of the left hand side of Equation 2 are polynomial in $\lambda$, the running time $\mathcal{T}_\mathcal{S}$ is also polynomial in $\lambda$.

## 4.2 Event E2

We consider two cases:

M1. The test session has a matching session.
M2. No party owns a matching session to the test session.

Events M1 and M2 are complementary. Therefore, if event (A3 $\wedge$ E2) occurs with non-negligible probability, then either (A3 $\wedge$ E2 $\wedge$ M1) or (A3 $\wedge$ E2 $\wedge$ M2) occurs with non-negligible probability.

### *4.2.1 Matching session exists*

We use $\mathcal{M}$ to construct an algorithm $\mathcal{S}$ that succeeds with non-negligible probability provided that the event (A3 $\wedge$ E2 $\wedge$ M1) occurs. In this case $\mathcal{S}$ establishes $n(\lambda)$ honest parties and randomly selects two sessions *sid* and *sid*$^*$. Suppose that $\mathcal{M}$ selects one of these sessions as the test session and the other as its matching session; if not then $\mathcal{S}$ aborts. The simulation follows the protocol description except for sessions *sid* and *sid*$^*$. For these sessions, owned by parties $\hat{A}$ and $\hat{B}$, $\mathcal{S}$ selects ephemeral private keys $\tilde{x}, \tilde{y} \in_R \{0,1\}^\lambda$, sets the ephemeral public keys as $U$ and $V$, thereby defining $\mathcal{H}_1(\tilde{x}, a) = u$ and $\mathcal{H}_1(\tilde{y}, b) = v$. Both sessions are assigned the same random session key $k$. The simulation fails if and only if $\mathcal{M}$ queries $\mathcal{H}_1$ with $(\tilde{x}, a)$ or $(\tilde{y}, b)$.

*Analysis.* With probability at least $2/s(\lambda)^2$ the adversary selects one of the two sessions *sid* and *sid*$^*$ as the test session and the other as its matching session. Without loss of generality, assume that $\hat{A}$ is the initiator of the test session. A successful adversary is allowed to query for at most one of the values in each pair $(\tilde{x}, a)$ and $(\tilde{y}, b)$. Suppose event E2 occurs. Then $\mathcal{M}$ issues *StaticKeyReveal*$(\hat{A})$ before a $(\tilde{x}, a)$ query to $\mathcal{H}_1$. Similarly, $\mathcal{M}$ issues *StaticKeyReveal*$(\hat{B})$ before a $(\tilde{y}, b)$ query to $\mathcal{H}_1$. Since $\tilde{x}$ and $\tilde{y}$ are used in only one session and $\mathcal{H}_1$ is a random function, $\mathcal{M}$ cannot obtain any information about $\tilde{x}$ and $\tilde{y}$ (except with negligible probability) without querying for them. Therefore if $\mathcal{M}$ selected the sessions with ephemeral public keys $U$ and $V$ as the test session and its matching session, the simulation does not fail. Suppose that event A3 occurs. Then a successful $\mathcal{M}$ queries $\mathcal{H}$ with $(\sigma, U, V, \hat{A}, \hat{B})$, where $\sigma$ is such that $\text{DDH}(UA^d, VB^e, \sigma) = 1$. To solve the CDH problem, $\mathcal{S}$ observes all $\mathcal{H}$ queries made by $\mathcal{M}$ of the form $(\sigma, U, V, \hat{A}, \hat{B})$; for every such query $\mathcal{S}$ checks if $\text{DDH}(UA^d, VB^e, \sigma) = 1$. When the DDH oracle returns 1, $\mathcal{S}$ computes $\text{CDH}(U, V) = \sigma G^{-abed} U^{-be} V^{-ad}$. The success probability of $\mathcal{S}$ is

$$\Pr(\mathcal{S}) \geq \frac{2}{s(\lambda)^2} p_2(\lambda), \tag{3}$$

where $p_2(\lambda)$ is the probability of event (A3 $\wedge$ E2 $\wedge$ M1). The simulation requires $\mathcal{S}$ to access the DDH oracle for (some of) the $\mathcal{H}$ queries made by $\mathcal{M}$, and therefore the running time of $\mathcal{S}$ satisfies

$$\mathcal{T}_{\mathcal{S}} \leq \big(\mathcal{T}_{\mathcal{G}} + (\mathcal{T}_{\mathrm{DDH}} + 2\mathcal{T}_{\mathcal{G}} + \mathcal{T}_{\mathcal{H}}) + \mathcal{T}_{\mathcal{H}_1} + \mathcal{T}_{\mathcal{H}_2}\big)\mathcal{T}_{\mathcal{M}}. \tag{4}$$

#### *4.2.2 Matching session does not exist*

We use $\mathcal{M}$ to construct an algorithm $\mathcal{S}$ that succeeds with non-negligible probability provided that the event (A3 $\wedge$ E2 $\wedge$ M2) occurs. Then $\mathcal{S}$ selects at random two distinct parties, which we call $\hat{A}$ and $\hat{B}$, and a session *sid*. Furthermore, $\mathcal{S}$ assigns static public key $V$ for $\hat{B}$, and random static key pairs for the remaining $n(\lambda) - 1$ parties. The adversary $\mathcal{M}$ is initiated on this set of parties. We suppose that $\mathcal{M}$ selects *sid* as the test session, and furthermore that $\hat{A}$ is the owner of *sid* and $\hat{B}$ its peer; if not $\mathcal{S}$ aborts. When $\mathcal{M}$ activates a party whose static private key $\mathcal{S}$ possesses, then $\mathcal{S}$ follows the protocol description. The only exception is the test session, for which $\mathcal{S}$ selects $\tilde{x} \in_R \{0,1\}^{\lambda}$, sets the ephemeral public key equal to $U$, and chooses a random key $k$ as the session key. Sessions owned by $\hat{B}$ are simulated as in Sect. 4.1.

*Analysis.* With probability at least $1/n(\lambda)^2$ $\mathcal{M}$ selects $\hat{A}$ as the owner and $\hat{B}$ as the peer of session *sid*. With probability at least $1/s(\lambda)$ $\mathcal{M}$ selects session *sid* as the test session. As in Sect. 4.2.1, the experiment may fail if $\mathcal{M}$ makes the query $(\tilde{x}, a)$ to $\mathcal{H}_1$. If event (A3 $\wedge$ E2) occurs, then $\mathcal{M}$ makes this query with only negligible probability. The freshness of the test session implies that $\mathcal{M}$ does not query *StaticKeyReveal*$(\hat{B})$, and therefore the experiment does not abort as described in item 7 of Sect. 4.1.

Without loss of generality assume that $\hat{A}$ is the initiator of the test session. Denote by $Y$ the incoming ephemeral public key selected by $\mathcal{M}$ for the test session. According to event A3 $\mathcal{M}$ queries $\mathcal{H}$ with $(\sigma, U, Y, \hat{A}, \hat{B})$, where $\mathrm{DDH}(UA^d, YB^e, \sigma) = 1$, in which case $\mathcal{S}$ computes

$$\Pi = \sigma Y^{-ad} V^{-ade} = G^{uve+uy}. \tag{5}$$

Without knowledge of $y$, $\mathcal{S}$ is unable to compute $\mathrm{CDH}(U, V)$. Following the approach of the Forking Lemma [19], $\mathcal{S}$ runs $\mathcal{M}$ on the same input and the same coin flips but with carefully modified answers to $\mathcal{H}_2$ queries. Note that $\mathcal{M}$ must have queried $\mathcal{H}_2$ with $(Y, \hat{A}, \hat{B})$ in its first run, because otherwise $\mathcal{M}$ would be unable to compute $\sigma$ except with negligible probability. For the second run of $\mathcal{M}$, $\mathcal{S}$ responds to $\mathcal{H}_2(Y, \hat{A}, \hat{B})$ with a value $e' \neq e$ selected uniformly at random. Another way of describing the second run is: $\mathcal{M}$ is rewound to the point where $\mathcal{M}$ queried $\mathcal{H}_2$ with $(Y, \hat{A}, \hat{B})$ and the query is answered with a random value $e'$ different from $e$. If $\mathcal{M}$ succeeds in the second run, $\mathcal{S}$ computes

$$\Pi' = \sigma' Y^{-ad'} V^{-ad'e'} = G^{uve'+uy}, \tag{6}$$

and thereafter obtains

$$\mathrm{CDH}(U, V) = \left(\frac{\Pi}{\Pi'}\right)^{(e-e')^{-1}}.$$

The forking is at the expense of introducing a wider gap in the reduction. The success probability of $\mathcal{S}$, modulo negligible terms, is

$$\Pr(\mathcal{S}) \geq \frac{1}{s(\lambda)} \frac{1}{n(\lambda)^2} \frac{C}{h_2(\lambda)} p_3(\lambda), \tag{7}$$

where $p_3(\lambda)$ is the probability of event (A3 $\wedge$ E2 $\wedge$ M2), and $C$ is a constant, which arises from the Forking Lemma. The running time analysis of the simulation is similar to the running time analysis in Sect. 4.1, except that $\mathcal{S}$ does not perform any additional checks for the $\mathcal{H}_1$ queries. Hence

$$\mathcal{T}_{\mathcal{S}} \leq \big(\mathcal{T}_{\mathcal{G}} + (\mathcal{T}_{\text{DDH}} + 2\mathcal{T}_{\mathcal{G}} + \mathcal{T}_{\mathcal{H}}) + \mathcal{T}_{\mathcal{H}_1} + \mathcal{T}_{\mathcal{H}_2}\big)\mathcal{T}_{\mathcal{M}}. \tag{8}$$

### 4.3 Analysis

Combining Eqs. 1, 3 and 7, the success probability of $\mathcal{S}$ is

$$\Pr(\mathcal{S}) \geq \max\left\{\frac{1}{n(\lambda)}p_1(\lambda), \frac{2}{s(\lambda)^2}p_2(\lambda), \frac{1}{s(\lambda)}\frac{1}{n(\lambda)^2}\frac{C}{h_2(\lambda)}p_3(\lambda)\right\}, \tag{9}$$

which is non-negligible in $\lambda$. Equations (2), (4) and (8) bound the running time of $\mathcal{S}$; taking the largest bound we obtain

$$\mathcal{T}_{\mathcal{S}} \leq \big(\mathcal{T}_{\mathcal{G}} + (\mathcal{T}_{\text{DDH}} + 2\mathcal{T}_{\mathcal{G}} + \mathcal{T}_{\mathcal{H}}) + (\mathcal{T}_{\mathcal{G}} + \mathcal{T}_{\mathcal{H}_1}) + \mathcal{T}_{\mathcal{H}_2}\big)\mathcal{T}_{\mathcal{M}}. \tag{10}$$

Thus, if $\mathcal{M}$ is polynomially bounded, then there is an $\mathcal{S}$ that succeeds in solving the GDH problem in $\mathcal{G}$ with non-negligible probability. Furthermore $\mathcal{S}$ runs in polynomial time, contradicting the GDH assumption in $\mathcal{G}$. This concludes the proof of Theorem 4.1. □

## 5 One-pass CMQV

In a nutshell, one-pass CMQV is two-pass CMQV, where the ephemeral public key $Y$ of the responder is the identity element in the group. To that end there is no need to include $Y$ in the key derivation.

**Definition 5.1** (*one-pass CMQV*) Party $\hat{A}$'s static private key is $a \in_R [1, q-1]$ and $\hat{A}$'s static public key is $A = G^a$. Similarly, $\hat{B}$'s static key pair is $(b, B)$. To establish a session key with $\hat{B}$, party $\hat{A}$ performs the following steps.

1. Select an ephemeral private key $\tilde{x} \in_R \{0,1\}^\lambda$.
2. Compute $x = \mathcal{H}_1(\tilde{x}, a)$ and the ephemeral public key $X = G^x$.
3. Compute $d = \mathcal{H}_2(X, \hat{A}, \hat{B})$, $\sigma = B^{x+da}$, and $k = \mathcal{H}(\sigma, X, \hat{A}, \hat{B})$.
4. Destroy $x$.
5. Send $(\hat{A}, \hat{B}, X)$ to $\hat{B}$.

Upon receiving $(\hat{A}, \hat{B}, X)$ party $\hat{B}$ does the following:

1. Verify that $X \in \mathcal{G}^*$.
2. Compute $d = \mathcal{H}_2(X, \hat{A}, \hat{B})$, $\sigma = (XA^d)^b$, and $k = \mathcal{H}(\sigma, X, \hat{A}, \hat{B})$.

The session key is $k$.

Even though the definition of secure protocol (Definition 2.2) does not depend on the number of protocol flows, the definition of fresh session has to be modified to fit the needs of a one-pass protocol. In particular, one-pass protocols cannot achieve wPFS since an adversary can compute a session key by learning the static private key of the responder.

**Definition 5.2** (*one-pass fresh session*) Let *sid* be the session identifier of a completed session, owned by an honest party $\hat{A}$ with intended peer $\hat{B}$, who is also honest. Let *sid*$^*$ be the session identifier of the matching session of *sid*, if it exists. Define *sid* to be *fresh* if none of the following conditions hold:

(i) $\mathcal{M}$ issues a *SessionKeyReveal*(*sid*) query or a *SessionKeyReveal*($sid^*$) query (if $sid^*$ exists);
(ii) if $\hat{A}$ is the initiator then $\mathcal{M}$ makes either of the following queries:
  – both *StaticKeyReveal*($\hat{A}$) and *EphemeralKeyReveal*(*sid*), or
  – *StaticKeyReveal*($\hat{B}$);
(iii) if $\hat{A}$ is the responder then $\mathcal{M}$ makes either of the following queries
  – *StaticKeyReveal*($\hat{A}$) or
  – *StaticKeyReveal*($\hat{B}$).

We point out that by replaying messages from $\hat{A}$ to $\hat{B}$ an adversary $\mathcal{M}$ could force multiple sessions owned by $\hat{B}$ sharing the same session key $k$. Let $S_k$ be the set of sessions owned by $\hat{B}$ with the same session key $k$. Since all sessions in $S_k$ have the same session identifiers, $\mathcal{M}$ cannot compromise a single session in $S_k$ without compromising all sessions in $S_k$. Therefore, the definition of session identifier accounts for replay attacks.

The security argument for one-pass CMQV is very similar to the security argument for two-pass CMQV (Sect. 4). As before it is enough to consider forging attacks and events E1 and E2. For event E1, the reduction requires only minor technical changes to account for the modified actions of the parties. For event E2, $\mathcal{S}$ selects two parties $\hat{A}$ and $\hat{B}$ that are given static public keys $U$ and $V$. Both $\hat{A}$ and $\hat{B}$ are simulated as parties whose static private keys $\mathcal{S}$ does not possess. The simulation of $\hat{A}$ and $\hat{B}$ is similar to the simulation of $\hat{B}$ in event E1. It is straightforward to verify that the simulation is perfect. Let *sid* be the test session identifier and assume that $\mathcal{S}$ made a correct guess for the communicating parties $\hat{A}$ and $\hat{B}$ and the test session selected by $\mathcal{M}$. Following the approach of Sect. 4.2.2 by rewinding $\mathcal{M}$ to the query $d = \mathcal{H}_2(Y, \hat{B}, \hat{A})$, where $Y$ is the ephemeral public key for the test session, $\mathcal{S}$ is able to compute

$$\mathrm{CDH}(U, V) = \left(\frac{\sigma}{\sigma'}\right)^{(d-d')^{-1}}.$$

If the owner of the test session, say $\hat{B}$, is also the initiator, then $\mathcal{S}$ knows the discrete logarithm of $Y$. In that case instead of rewinding the adversary, $\mathcal{S}$ can compute $\mathrm{CDH}(U, V) = (\sigma U^{-y})^{d^{-1}}$.

## 6 Concluding remarks

The paper presented CMQV, a modification of the MQV key agreement protocol. On the positive side the new protocol is secure in the extended Canetti-Krawzcyk model. Moreover it achieves the performance of the original MQV protocol, and has intuitive design principles and a relatively simple security proof. On the negative side the reduction argument is not tight, in particular the Forking Lemma appears to be essential for the security argument. It remains to be seen if there exists a protocol that achieves the performance of MQV and at the same time enjoys a security reduction that is as tight as the security reduction for NAXOS.

**Acknowledgments** This paper owes much to the suggestions, help and advice of Alfred Menezes. I would also like to thank Hugo Krawczyk and the two anonymous referees for their valuable comments.

## References


1. Antipa A., Brown D., Menezes A., Struik R., Vanstone S.: Validation of elliptic curve public keys. Public Key Cryptography – PKC 2003, LNCS, vol. 2567, pp. 211–223 (2003).
2. Bellare M., Palacio A.: The knowledge-of-exponent assumptions and 3-round zero-knowledge protocols. Advances in Cryptology – CRYPTO 2004, LNCS, vol. 3152, pp. 273–289 (2004).
3. Bellare M., Pointcheval D., Rogaway P.: Authenticated key exchange secure against dictionary attacks. Advances in Cryptol. – EUROCRYPT 2001, LNCS, vol. 1807, pp. 139–155 (2001).
4. Bellare M., Rogaway P.: Entity authentication and key distribution. Advances in Cryptol. – CRYPTO '93, LNCS, vol. 773, pp. 110–125 (1993).
5. Blake-Wilson S., Menezes A.: Unknown key-share attacks on the station-to-station STS protocol. Public Key Cryptography – PKC '99, LNCS, vol. 1560, pp. 154–170 (1999).
6. Canetti R., Krawczyk H.: Analysis of key-exchange protocols and their use for building secure channels. Advances in Cryptology – EUROCRYPT 2001, LNCS, vol. 2045, pp. 453–474, Full version available at http://eprint.iacr.org/2001/040 (2001).
7. Choo K-K., Boyd C., Hitchcock Y.: Examining indistinguishability-based proof models for key establishment protocols. Advances in Cryptology – ASIACRYPT 2005, LNCS, vol. 3788, pp. 585–604 (2005).
8. Krawczyk H.: HMQV: A high-performance secure Diffie-Hellman protocol. Advances in Cryptology – CRYPTO 2005, LNCS, vol. 3621, pp. 546–566, Full version available at http://eprint.iacr.org/2005/176 (2005).
9. Krawczyk H.: HMQV in IEEE P1363. submission to the IEEE P1363 working group, July 2006, http://grouper.ieee.org/groups/1363/P1363-Reaffirm/submissions/krawczyk-hmqv-spec.pdf.
10. Kunz-Jacques S., Poincheval D.: About the security of MTI/C0 and MQV. Security and Cryptography for Networks – SNC 2006, LNCS, vol. 4116, pp. 156–172 (2006).
11. LaMacchia B., Lauter K., Mityagin A.: Stronger security of authenticated key exchange. ProvSec 2007, LNCS, vol. 4784, pp. 1–16, Preliminary version available at http://eprint.iacr.org/2006/073(2007).
12. Lauter K., Mityagin A.: Security analysis of KEA authenticated key exchange. Public Key Cryptography – PKC 2006, LNCS, vol. 3958, pp. 378–394 (2006).
13. Law L., Menezes A., Qu M., Solinas J., Vanstone S.: An efficient protocol for authenticated key agreement. Des. Codes Cryptogr. **28**, 119–134 (2003).
14. Lim C., Lee P.: A key recovery attack on discrete log-based schemes using a prime order subgroup. Advances in Cryptology – CRYPTO '94, LNCS, vol. 1294, pp. 249–263 (1994).
15. Menezes A.: Another look at HMQV. J. Math. Cryptol. **1**, 148–175 (2007).
16. Menezes A., Ustaoglu B.: On the importance of public-key validation in the MQV and HMQV key agreement protocols. Progress in Cryptology – INDOCRYPT 2006, LNCS, vol. 4329, pp. 133–147 (2006).
17. Menezes A., van Oorschot P., Vanstone S.: Handbook of Applied Cryptography. CRC Press, Boca Raton, Florida, USA (1997).
18. Okamoto T., Pointcheval D.: The Gap-Problems: a new class of problems for the security of cryptographic schemes. Public Key Cryptography – PKC 2001, LNCS, vol. 1992, pp. 104–118 (2001).
19. Poincheval D., Stern J.: Security arguments for digital signatures and blind signatures. J. Cryptol. **13**(3), 361–396 (2000).