An extended abstract of this work appears in *Proceedings of the 5th International Conference (SCN '06)*
(6 – 8 september 2006, Maiori, Italy)
M. Yung Ed., Springer-Verlag, LNCS 4116, Pages 186–200.

A New Key Exchange Protocol Based on MQV Assuming Public Computations

Sébastien Kunz-Jacques[1,2] and David Pointcheval[1]

[1] École normale supérieure, 45 rue d'Ulm, 75005 Paris, France
David.Pointcheval@ens.fr
[2] DCSSI Crypto Lab, 51 boulevard de La Tour-Maubourg
F-75700 Paris 07 SP, France
kunzjacq@yahoo.fr

Abstract. Designing authenticated key exchange algorithms is a problem well understood in cryptography: there are established security models, and proposals proved secure in these models. However, models currently used assume that a honest entity involved in a key exchange is trusted as a whole. In many practical contexts, the entity is divided in an *authentication device* storing a private key and having low computing power, and a *computing device*, that performs part of the computations required by protocol runs. The computing device might be a PC connected to the Internet, and the authenticating device a smart card. In that case as well in many others, a compromise of the computing device is to be expected. We therefore propose a variant of the MQV and HMQV key exchange protocols secure in that context, unlike the original protocols. The security claim is supported by a proof in a model derived from the Canetti-Krawczyk one, which takes into account more general rogue behaviours of the computing device.

1 Introduction

Key exchange, together with other basic primitives like encryption and signature, constitutes a building block of modern cryptography. Key exchange algorithms enable two parties communicating on an insecure channel to agree on a common secret value. The Diffie-Hellman algorithm [7] was the first key-exchange algorithm not requiring a pre-shared static secret between the parties. It does not however enforce parties authentication, and is therefore vulnerable to man-in-the-middle attacks. After that seminal paper, many authenticated key-exchange (AKE) protocols were proposed, some of them with security proofs.

Current security models for key exchange [4, 3, 5, 6] take into account active attackers, and model the secrecy of session keys together with the mutual authentication property, that is, the assurance for each participant of a protocol run that it talks to whom it *thinks* it is talking to.

Our purpose is to motivate a new attack scenario that arises naturally when implementing an AKE protocol. We thus define a security model including this attack capability and then build a protocol secure given this new constraint. We namely focus on situations where it is convenient to split an entity performing a run of a key exchange protocol into an *authentication device* and an untrusted *computing device*. The authenticating device enforces the confidentiality of the authentication data while some computing operations required by the protocol are carried out by the computing device. This allows to use an authentication device with little computing power, and to make computing devices independent from users.

In such a framework, an AKE protocol is expected to mitigate the consequences of a computing device compromise. Specifically, an attacker that had the opportunity to interact offline with some authentication devices should not be able to authenticate itself in subsequent protocol runs.

Several applications might benefit from an AKE protocol able to cope with a computing device. Mobile phones include smart cards which store the user authentication data; the handsets themselves are the computing devices. PCs equipped with a crypto token have a lot more computing power than the token itself, but may be plagued by spyware or viruses. New designs can also be devised. For example, using an AKE protocol secure in our model, one could build authenticated and escrowable end-to-end encrypted communications in mobile phone networks: the handsets act as the authentication devices and the mobile phone base stations as the computing devices. Session keys are negotiated between two handsets when a communication is initiated. With such a setup, the network operator knows session keys and can therefore decipher calls as required by the law, but *is still unable to fake the users authentication.*

The security model we define takes into account the capacity for an attacker to compromise a computing device, in the following strong sense: the attacker performs itself the operations normally assumed by the computing device, and therefore interacts with the authenticating device as a computing device would do. In that situation, the attacker can compute session keys, but the the model requires that after an arbitrary number of such interactions, the attacker is unable to fake the identity of the authenticating device it interacted with. The model is named the *public computation model*, because the attacker has both passive and active access to the computation devices.

MQV, a well-known signature-less AKE protocol, is a good candidate to build a protocol that is secure in the public computation model, although it was not designed to take into account such threats. We show that MQV itself [15, 24], or its variant HMQV [11], are not secure if scalar multiplications are moved into the computing device; however, only slight changes are required to make MQV secure in that setting. We present a 4-pass variant of MQV that is provably secure in our model. The security proof assumes the difficulty of the CDH problem and lies in the random oracle model.

Related Work. The public computation model is analogous to the Canetti-Krawczyk model [5]. In the latter, access to the computing device would have been granted through "Session State Queries". We chose what seemed a simpler path to remove the computing device altogether and allow the attacker to interact freely with authentication devices. Instead of performing Session State queries, the attacker therefore assumes the role the computing device itself. This allows not only access to values in an honest computing device, but also arbitrary man-in-the-middle scenarios where the computing device behaves abnormally to gain information about long-term secrets.

In [11], all variants of HMQV including the 3-pass variant HMQV-C are proved secure in the CK model, *assuming intermediate scalar values are stored in protected memory*, that is, out of reach of Session State queries. This is not very satisfactory as these values are inherently ephemeral. The protocol we propose is designed to overcome this drawback. Our security proof result can therefore be seen as an extension of what is proved in [11].

Contrary to [11], it is assumed that the protocol uses a *prime-order* group. We do not pretend to eliminate the need for subgroup membership tests when a non prime-order group is used. As shown in [14], these membership issues can cause subtle errors in security proofs.

The access of the attacker to the authentication device is very much like the Access queries of [23], which addresses a problem similar to ours for the Needham and Shroeder protocol.

Overall, the model introduced is a very classical Real-or-Random one. It is built around two natural notions for key exchange protocols, real partnership and intended partnership: they designate respectively the relationship between users really exchanging messages and intending to exchange messages. Real partners are "peers" while intended partners are "assumed peers" in the CK model; the "real partnership" relation comes from the "session IDs" of [4]. To slice proofs into more palatable parts, properties relevant to un-authenticated key exchange and authenticated key exchange are covered by two separate security games. As a side effect, it is very simple to specialize the model to un-authenticated protocols.

The introduction of a "computing device" in key exchange is analogous to some works on "server-aided computations" aimed at improving the efficiency of RSA signatures [12, 2]. While the resulting protocols were proved insecure [21, 18, 19], our protocol proposal uses a mix of external help (the computing device) and use-and-throw coupons to avoid computing scalar multiplications. Coupons are a well-known trick to improve the efficiency of discrete log based signature schemes [17], which MQV is related to. We do not provide a general solution for relying on an external device to compute scalar multiplications like in [8], but rather provide an ad-hoc solution tailored to MQV. [8] could be used to eliminate completely the need for coupons in our variant of MQV, however it requires to independent computing units in the computing device, an hypothesis which cannot be easily verified by the authenticating device.

Paper Outline. The paper is structured as follows. First, we review some general concepts related to the security of key exchange protocols in section 2. In section 3, we introduce the public computation model. We then review MQV in section 4.1 and explain why its natural implementation with both an authentication device and a computing device cannot be secure in that model. The variant of MQV in the public computation model, MQV-p, is presented in paragraph 4.2. The algorithmic problems used in the proof and the proof results are summarized in section 5. A sketch of proof outlining the motivations of some choices of proof techniques is then provided section 6. The proof itself is laid out appendix A.

2 Security Goals and Related Concepts

We build the security notions for key exchange around the concepts of intended partner and real partner. The real partner of a user U is the user who receives (resp. sends) the messages sent (resp. received) by U; it is not necessarily known by U. On the other hand, the intended partner of U, which is defined only in the authenticated case, is the user U thinks it is talking to. The security of an unauthenticated key exchange protocol is then expressed as follows: the key resulting from a key exchange must be secret for anybody except U and its real partner. This is the *semantic security* property. An AKE protocol must further satisfy the *mutual authentication* (MA) requirement: a protocol run should complete successfully for some user U only if the *intended partner* of U matches its real partner. In an AKE protocol, the combination of semantic security and mutual authentication yields the property that a key computed after a protocol run completed successfully is shared with the intended partner, and with it only. To define rigorously these security goals, we need to take a closer look at some general concepts related to authenticated key exchange.

2.1 Session Identifiers

Throughout the paper, we need to put labels on protocol runs. Formally, we could use a "global" naming scheme, where each run is uniquely identified throughout all runs performed by all users. This would not have a concrete meaning however, since users only know about the protocol runs they perform themselves. Therefore we identify protocol runs by user - session index pairs. This way, we can assume that each user U engaged in a session (U, s) knows about s.

2.2 Key Material, Session View, Real Partners

For any KE protocol, the moment when a user has enough information to compute the session key can be defined. In the protocols we consider, we represent this by a flag, $\mathsf{KeyMaterialReceived}$, that is set to true when the session key can be computed. The data exchanged required to compute the session key are called the *key material*.

For some session (U, s), we denote by $\mathsf{View}(\mathsf{U}, \mathsf{s})$ all the messages sent and received by U during session s before $\mathsf{KeyMaterialReceived} = \mathsf{true}$, described in a user-independent way. $\mathsf{View}(\mathsf{U}, \mathsf{s})$ is only defined when U has set $\mathsf{KeyMaterialReceived}$ to true in session s. The session key is computed by U as a function of its View and its private key. When $\mathsf{View}(\mathsf{U}, \mathsf{s}) = \mathsf{View}(\mathsf{U'}, \mathsf{s'})$, we say that sessions (U, s) and $(\mathsf{U'}, \mathsf{s'})$ *match*, and that U and $\mathsf{U'}$ are *real* partners for sessions s and $\mathsf{s'}$.

2.3 Intended Partners, Key Acceptance and Mutual Authentication

In an honest protocol run of an authenticated key exchange algorithm, each user has an intended partner. The intended partner is defined in a protocol-dependent way but it must be possible to express it as a function of the messages exchanged during a session and of the public key of the user. This function might not be easily computable; this happens for example in the case of a protocol including some identity hiding functionality, like -I and -R variants of SIGMA [10]. When the way to derive the intended partner identity is not obvious, it should be clearly stated in a protocol description. Except in these special cases, it is usually straightforward to define the identity of the intended partner in terms of the messages exchanged during a protocol run.

During an AKE protocol run, a user acknowledges at some point that it is talking to its intended partner. In the protocols we describe, this is again materialized by a flag, $\mathsf{KeyAccepted}$. Therefore a user state in an AKE protocol is defined by the Boolean values $\mathsf{KeyMaterialReceived}$ and $\mathsf{KeyAccepted}$. The mutual authentication property of an AKE protocol can now be easily defined: a protocol has the MA property if whenever a session (U, s) completes successfully ($\mathsf{KeyAccepted} = \mathsf{true}$), user U has a real partner $\mathsf{U'}$ for session s, and it is equal to its intended partner. Note that for $\mathsf{U'}$ to be U's real partner for some session $\mathsf{s'}$, $\mathsf{View}(\mathsf{U'}, \mathsf{s'})$ must be defined, which implies that $\mathsf{KeyMaterialReceived} = \mathsf{true}$ for session $(\mathsf{U'}, \mathsf{s'})$.

3 Security Model

We define in this section our AKE security model. The security goals are formalized into games between an attacker E and a simulator S running instances of an AKE protocol between several users. In these games, E directs the users actions regarding executions of the protocol, and has total control over the messages exchanged between the users.

E's capabilities correspond to *queries* that it can make to S. Queries are listed in section 3.1. In particular, E can get the long-term private keys corresponding to legitimate identities in two ways: it can obtain keys of existing users controlled by S through $\mathsf{Corrupt}$ queries, or register its own users through $\mathsf{Register}$ queries. The public computation model therefore allows the presence of users controlled by the attacker alongside the ones controlled by S. In particular, attacks requiring to dynamically register a public key, like the one of Kaliski on MQV [9], are within the scope of the model.

S *does not* stop simulating a corrupted user. A user that was targeted by a $\mathsf{Corrupt}$ query can therefore be simulated by S *and* impersonated by E. Therefore a corrupted user might still be involved in honest protocol runs, managed by S. We name them *honest* sessions and session views.

The new attack scenario that we take into account translates into two new queries available to E, $\mathsf{IniAuth}$ and $\mathsf{SendAuth}$, enabling it to interact freely with the authentication device of any user.

The model is composed of two games: a real-or-random game G_{ror} modeling the secrecy of the negotiated key, as explained in section 3.3; and a game G_{ma} where the goal of the attacker is to break mutual authentication, defined in section 3.4.

Differences between the public computation model and the Canetti-Krawczyk model [5] are as follows:

- Each user is split in two parts: an authentication device and a computing device. This corresponds somewhat to the "protected" and "unprotected" memory in the CK model, but is more flexible because the attacker can impersonate a computing device in an arbitrary way instead of only be granted access to the memory of honest computing devices.
- Security notions for un-authenticated and authenticated key exchange protocols are modelled by two separate security games.
- Key secrecy is modeled by a real-or-random game instead of the find-then-guess game of [5, 4]. The two corresponding security properties are equivalent, but there is a loss factor linear in the number of sessions from the find-then-guess game to the real-or-random game [1] in the security bounds obtained.

3.1 Simulation and Attacker's Queries

E can issue different queries to S to control sessions and messages exchanged by users. It is also given complete control over messages between users: messages that are supposed to be sent by users in the real protocol are actually handed over by S to E, along with the corresponding session identifier. E can send messages to users through Send queries: $\mathsf{Send}(\mathsf{U}, \mathsf{s}, M)$ sends message M to user U as part of session s.

New sessions are opened through $\mathsf{Initiate}$ queries: $\mathsf{s} = \mathsf{Initiate}(\mathsf{ID}_{\mathsf{U}}, \mathsf{ID}_{\mathsf{U}'})$ tells user U to initiate a new session with user U'. U is therefore the initiating user of the session. The attacker is answered a session identifier s that is later used in Send queries. ID_{U} and $\mathsf{ID}_{\mathsf{U}'}$ must match registered identities of users either simulated by S or created by E through $\mathsf{Register}$ queries as described below.

When some user U' controlled by the simulator receives a message that does not belong to an existing session, and that can be interpreted as the first message of a new session, it creates a new session identifier s' that is handed over to E.

E's attack capabilities are modeled by the following queries:

– $\mathsf{Corrupt}(\mathsf{U})$: obtain the long-term private key of U.
– $\mathsf{Register}(k, \mathsf{ID}_\mathsf{V})$: register public key k for identity ID_V. The public key may have already been assigned to some user, however ID_V must not match the identity of an existing user. ID_V is the identity of a new legitimate user V controlled by the attacker. Remark that the model *does not require* the CA to ask for proofs of knowledge of the private keys during identity registration.
– $\mathsf{t} = \mathsf{IniAuth}(\mathsf{U})$ and $\mathsf{SendAuth}(\mathsf{t}, M)$: these two queries mimic $\mathsf{Initiate}$ and Send queries and model E's access to authentication devices. An "authentication session index" t is used to allow and to keep track of concurrent authentication sessions. In a signed Diffie-Hellman protocol for instance, $\mathsf{SendAuth}(\mathsf{t}, M)$ would simply return the signature of message M by user U if the authentication session t has been opened for user U.

$\mathsf{Corrupt}$ queries model long-term key material leakage; $\mathsf{Register}$ queries model users that are created by E, for example when E chooses a public key depending on some observed data. $\mathsf{IniAuth}$ and $\mathsf{SendAuth}$ queries model the access to the authentication device.

3.2 Common Framework for Security Games

In the two games G_{ma} and G_{ror}, S simulates real protocol sessions according to the queries made by E as in section 3.1. The simulations used in these games differ only by the value handed over to E when a session completes successfully ($\mathsf{KeyAccepted} \leftarrow \mathsf{true}$): in game G_{ma}, nothing is given to E whereas in game G_{ror}, a "real-or-random" value is revealed.

3.3 Semantic Security Game G_{ror}

The real-or-random game G_{ror} models the key secrecy in front of passive attacks. E wins that game if it manages to distinguish real session keys from random values.

In that game, S first draws a global random bit b. This bit decides whether real session keys or random values are to be revealed to E, whose goal is to guess b correctly. If $b = 1$, a simulation $\mathcal{S}^{\mathsf{Real}}$ is used: the real session key is revealed to E after a session completes successfully. If $b = 0$, the simulation $\mathcal{S}^{\mathsf{Random}}$ is performed as follows: first, S sets up a private random oracle H_0. Next, S simulates protocol runs as in $\mathcal{S}^{\mathsf{Real}}$. When a session (U, s) completes successfully with at least one honest matching session $(\mathsf{U'}, \mathsf{s'})$, the value revealed to E is equal to $\mathsf{H}_0(\mathsf{View}(\mathsf{U}, \mathsf{s}))$ (remember that a honest session is a session simulated by S, irrespectively of whether the corresponding user was corrupted or not.) If the session completes successfully without a honest matching session, the real key is revealed, as E might be in a position to compute it, for example because it impersonated U's real partner for session s.

Let b' be E's answer. E's *advantage* in game G_{ror} is

$$\mathsf{Adv}_{\mathsf{ror}} = \left| \mathsf{P}_{b=1}[b' = 1] - \mathsf{P}_{b=0}[b' = 1] \right| .$$

3.4 Mutual Authentication Game G_{ma}

Game G_{ma} models the mutual authentication property of the protocol which, together with the key secrecy property from game G_{ror}, guarantees the resistance to active attacks of an AKE protocol. In game G_{ma}, E's goal is to get some user U to end successfully some session s ($\mathsf{KeyAccepted}(\mathsf{U},\mathsf{s}) = \mathsf{true}$) while its real partner differs from its intended partner U' (including while U has no real partner for that session.) The targeted session is called the attacked session.

To succeed, E must additionally not perform

– a SendAuth query targeted at the authenticating device of U' (that is, a SendAuth query on an authentication session t opened with IniAuth(U'))
– a Corrupt query on U or U'

between the beginning of the attacked session and the moment when U set KeyAccepted to true in the attacked session.

Since Corrupt queries on the user involved in the attacked session and its intended partner are banned, the model does not take into account key-compromise impersonation attacks.

3.5 The Public Computation Model in the Un-authenticated Setting

Our model can be adapted very simply to the un-authenticated setting. In that context, only key secrecy can be expected. In an un-authenticated protocol a key is accepted as soon as it can be computed; in our formalism, KeyAccepted is by definition equal to KeyMaterialReceived. Key secrecy is then modelled by game G_{ror} alone.

4 MQV Revisited with Public Computations in Sight: MQV-p

The MQV protocol with key confirmation is a well-known authenticated key-exchange algorithm that is forward-secure and that does not use signatures. It can be compared with some variants of the MTI protocols [13, 16]. One of the distinctive features of MQV is however to allow for an efficient split of its implementation between an authentication device and a computing device, as described in the introduction.

Key Confirmation. We consider the MQV variant that includes a key confirmation round: a key is used only if a value proving that the other party has managed to compute the session key, the *confirmation key*, has been received from the other party. With the conventions of paragraph 2.3, KeyAccepted is set to true only when the correct key confirmation has been received.

Key Derivation Function. In our protocol descriptions, KDF is a key derivation function: $\mathsf{KDF}(i, S)$ derives a key k_i from some secret s. It can classically be constructed from PRFs and universal hash functions (this is folklore; see for example [20].) In the proofs, KDF is modeled as a random oracle H_1: $\mathsf{KDF}(i, s) = \mathsf{H}_1(i||s)$.

4.1 The MQV Protocol

Let G be a large subgroup of an elliptic curve group over a finite field. G is assumed to be of prime order p, and P is a generator of G. The private key sk_U of a user U is an element of $\mathbb{Z}_p$ and the corresponding public key pk_U is $\mathsf{sk}_\mathsf{U}P$. For $Q \in G$, $T(Q)$ is the lower half of the x-coordinate of Q. The MQV protocol is depicted on figure 1.

Intended Partner Definition. The initiator of a session, Alice, knows who it intends to talk to before starting a protocol run; therefore the intended partner of Alice is fixed before the session starts. The intended partner of the responder Bob is set to the identity received in the first message.

MQV with Public Computations. The natural way to split authentication data storage and computations in MQV is as follows (see figure 2):

- The authentication device for user U computes pairs $(rP, r+T(rP)s_{\mathsf{U}})$ for random $r \in \mathbb{Z}_p$;
- For each protocol run, the computing device requests such a pair (R, s), sends R to the other party, receives R', and computes the key material $\mathsf{KM} = s(R' + T(R')P_{\mathsf{U}'})$.

In this description, it seems that the authentication device has to perform the scalar multiplication $r \rightarrow rP$ which is a costly operation. However, this can be practically avoided by pre-computing and storing in the authentication device pairs (r, rP) or even the authentication pairs $(rP, r + T(rP)s_{\mathsf{U}})$ themselves.

This approach however has one major security shortcoming: if a corrupted computing device stores a valid authentication pair $(rP, r+T(rP)s_{\mathsf{U}})$, it can authenticate as user U indefinitely without interacting with the authentication device anymore. This is clearly not desired and is a violation of the mutual authentication property in our security model; see section 3.4.

HMQV. The same difficulty arises with HMQV which uses $\mathsf{H}(R||\mathsf{ID}_{\mathsf{U}'})$ instead of $T(R)$, where U' is the intended partner of U. In any case, the authentication information is static and can be reused indefinitely.

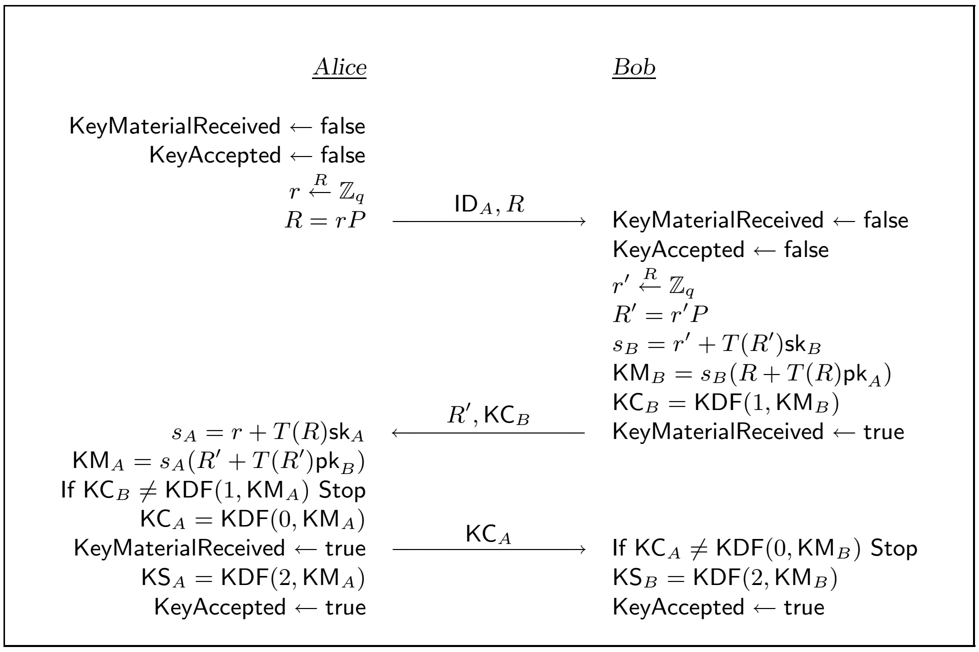
Alice
Bob
KeyMaterialReceived ← false
KeyAccepted ← false
$r \overset{R}{\leftarrow} \mathbb{Z}_q$
$R = rP$
ID_A, R
KeyMaterialReceived ← false
KeyAccepted ← false
$r' \overset{R}{\leftarrow} \mathbb{Z}_q$
$R' = r'P$
$s_B = r' + T(R')\mathsf{sk}_B$
$\mathsf{KM}_B = s_B(R + T(R)\mathsf{pk}_A)$
$\mathsf{KC}_B = \mathsf{KDF}(1, \mathsf{KM}_B)$
R', KC_B
KeyMaterialReceived ← true
$s_A = r + T(R)\mathsf{sk}_A$
$\mathsf{KM}_A = s_A(R' + T(R')\mathsf{pk}_B)$
If $\mathsf{KC}_B \neq \mathsf{KDF}(1, \mathsf{KM}_A)$ Stop
$\mathsf{KC}_A = \mathsf{KDF}(0, \mathsf{KM}_A)$
KeyMaterialReceived ← true
KC_A
If $\mathsf{KC}_A \neq \mathsf{KDF}(0, \mathsf{KM}_B)$ Stop
$\mathsf{KS}_A = \mathsf{KDF}(2, \mathsf{KM}_A)$
$\mathsf{KS}_B = \mathsf{KDF}(2, \mathsf{KM}_B)$
KeyAccepted ← true
KeyAccepted ← true

Fig. 1. An honest execution of the plain MQV protocol

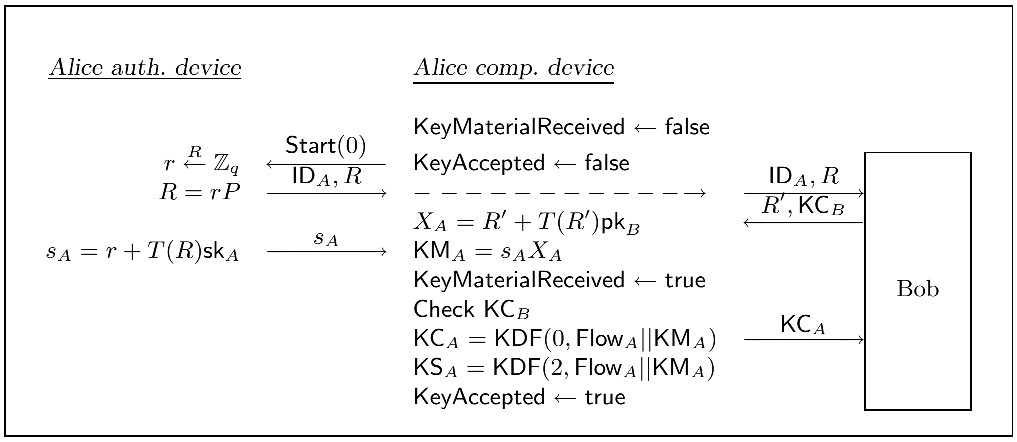
Alice auth. device
Alice comp. device
$r \xleftarrow{R} \mathbb{Z}_q$
$R = rP$
$s_A = r + T(R)\mathsf{sk}_A$
Start(0)
ID_A, R
s_A
KeyMaterialReceived ← false
KeyAccepted ← false
$X_A = R' + T(R')\mathsf{pk}_B$
$\mathsf{KM}_A = s_A X_A$
KeyMaterialReceived ← true
Check KC_B
$\mathsf{KC}_A = \mathsf{KDF}(0, \mathsf{Flow}_A || \mathsf{KM}_A)$
$\mathsf{KS}_A = \mathsf{KDF}(2, \mathsf{Flow}_A || \mathsf{KM}_A)$
KeyAccepted ← true
ID_A, R
R', KC_B
KC_A
Bob

Fig. 2. Plain MQV honest execution with an authentication token, point of view of Alice

4.2 MQV-p

The main idea to enhance MQV is to introduce some variability that cannot be controlled by the user in authentication tokens. This is performed by making them depend on the random point of the partner. In MQV-p, the truncation function $T(\cdot)$ is therefore replaced by a hash function H whose input consists in the identities of both users and the two exchanged random points. Unfortunately, applying this simple change to MQV leads to a three-pass protocol whose proof seems difficult even in the random oracle model; indeed, in this three-pass protocol, the adversary has partial control on input values of the oracle whose output we need to program. Because of this, we are unable to simulate sessions correctly when some private keys are replaced by challenges (see section A.5.) There are several ways to overcome this issue, either using a random value revealed later in the protocol, or making the responding user commit its random point first. These two techniques lead to a 4-pass protocol. Using a random value, one obtains the scheme described figure 3. In MQV-p, intended partner identities are derived as in the MQV protocol.

5 Algorithmic Hypotheses and Security Results

5.1 CDH Problem

On input $(xP, yP) \in_R G^2$, output xyP in G. $\mathsf{Succ}_{\mathsf{CDH}}(t, G)$ is the maximum success rate of an attacker against CDH running in time t in group G.

5.2 HCDH problem

HCDH depends on a random oracle H' having a h-bit output size. On input $(X = xP, Y) \in_R G^2$, an answer to this HCDH instance is a pair

$$(R', x(R' + \mathsf{H}'(R')Y)) \in G^2. \tag{1}$$

$\mathsf{Succ}_{\mathsf{HCDH}}(t, G, q_{\mathsf{H}'})$ is the maximum success rate of an attacker against HCDH running in time t in group G, and making at most $q_{\mathsf{H}'}$ H'- queries.

In appendix B, an adversary against HCDH is transformed into one against CDH by a classical splitting lemma argument, yielding the following inequality:

$$\begin{array}{rcl}
\underline{\textit{Alice}} & & \underline{\textit{Bob}} \\
\mathsf{KeyMaterialReceived} \longleftarrow \mathsf{false} & & \\
\mathsf{KeyAccepted} \longleftarrow \mathsf{false} & & \\
z \overset{R}{\longleftarrow} \{0,1\}^u & & \\
r \overset{R}{\longleftarrow} \mathbb{Z}_q & & \\
R = rP & \xrightarrow{\mathsf{ID}_A, R} & \mathsf{KeyMaterialReceived} \longleftarrow \mathsf{false} \\
 & & \mathsf{KeyAccepted} \longleftarrow \mathsf{false} \\
 & & r' \overset{R}{\longleftarrow} \mathbb{Z}_q \\
\mathsf{Flow}_A = \{\mathsf{ID}_A, R, \mathsf{ID}_B, R'\} & \xleftarrow{R'} & R' = r'P \\
s_A = r + \mathsf{H}(0||z||\mathsf{Flow}_A)\mathsf{sk}_A & & \\
\mathsf{KM}_A = s_A(R' + \mathsf{H}(1||\mathsf{Flow}_A)\mathsf{pk}_B) & & \\
\mathsf{KC}_A = \mathsf{KDF}(0, \mathsf{Flow}_A||\mathsf{KM}_A) & & \\
\mathsf{KeyMaterialReceived} \longleftarrow \mathsf{true} & \xrightarrow{z, \mathsf{KC}_A} & \mathsf{Flow}_B = \{\mathsf{ID}_A, R, \mathsf{ID}_B, R'\} \\
 & & s_B = r' + \mathsf{H}(1||\mathsf{Flow}_B)\mathsf{sk}_B \\
 & & \mathsf{KM}_B = s_B(R + \mathsf{H}(0||z||\mathsf{Flow}_B)\mathsf{pk}_A) \\
 & & \text{If } \mathsf{KC}_A \neq \mathsf{KDF}(0, \mathsf{Flow}_B||\mathsf{KM}_B) \text{ Stop} \\
 & & \mathsf{KeyMaterialReceived} \longleftarrow \mathsf{true} \\
\text{If } \mathsf{KC}_B \neq \mathsf{KDF}(1, \mathsf{Flow}_A||\mathsf{KM}_A) \text{ Stop} & \xleftarrow{\mathsf{KC}_B} & \mathsf{KC}_B = \mathsf{KDF}(1, \mathsf{Flow}_B||\mathsf{KM}_B) \\
\mathsf{KS}_A = \mathsf{KDF}(2, \mathsf{Flow}_A||\mathsf{KM}_A) & & \mathsf{KS}_B = \mathsf{KDF}(2, \mathsf{Flow}_B||\mathsf{KM}_B) \\
\mathsf{KeyAccepted} \longleftarrow \mathsf{true} & & \mathsf{KeyAccepted} \longleftarrow \mathsf{true}
\end{array}$$

Fig. 3. An honest execution of 4-pass MQV-p

$$\frac{[\mathsf{Succ}_{\mathsf{HCDH}}(2t, G, q_{\mathsf{H}'}) - 2/p - 2^{-h}]^2}{4q_{\mathsf{H}'}} - 2^{-h} \leq \mathsf{Succ}_{\mathsf{CDH}}(t, G). \qquad (2)$$

5.3 Security Bounds for Game G_{ror} and G_{ma}

In the two next theorems, h is the bit-size of the output of H, k is the one of KDF, u is the bit size of z, n_u is the maximal number of users simulated, p is the (prime) order of G, and t_{exp} is the scalar multiplication computation time[1].

Theorem 1 *If H is modeled as a random oracle, an adversary against the privacy of the session key of MQV-p in a group G, running in time t and making at most q_{KDF} KDF-queries, and q_s Initiate-queries, has advantage at most*

$$\mathit{Adv}_{ror}(t, G, q_{KDF}, q_s) \leq q_{KDF}\, q_s \left[\mathit{Succ}_{CDH}(t + (2q_s + 1)t_{exp}, G) + \frac{2q_s^2}{p} \right].$$

Theorem 2 *If H is modeled as a random oracle, an adversary against the mutual authentication of MQV-p in a group G, running in time t and making at most q_H H-queries, q_{KDF} KDF queries, and q_s Initiate-queries, has advantage at most*

$$\mathit{Succ}_{ma}(t, G, q_H, q_{KDF}, q_s) - 2^{-k} \leq (2\, q_s\, n_u^2\, \beta + q_{KDF})\, \mathit{Succ}_{HCDH}(t + 2q_s t_{exp}, G)$$

with $\beta = [(1 - q_{KDF}2^{-k})(1 - q_H/p)(1 - q_H \max\{2^{-u}, 1/p\})]^{-1}$.

[1] The time to compute a sum of k scalar multiplications, $a_1, P_1, \ldots, a_k, P_k \rightarrow \sum a_i P_i$ is assumed to be equal to t_{exp} if k is small. This is the case if Shamir's trick is used.

6 MQV-p Security: Sketch of Proof

We are looking for an upper bound for E's advantage in game G_{ror} and E's success rate in game G_{ma}. H and KDF are modeled as random oracles.

6.1 Game G_{ror}

We want to bound $|\mathsf{P}_{\mathcal{S}^{\mathsf{Real}}}[b' = 1] - \mathsf{P}_{\mathcal{S}^{\mathsf{Random}}}[b' = 1]|$, where b' is E's answer, in the context of a passive attack

The idea is to modify $\mathcal{S}^{\mathsf{Real}}$ and $\mathcal{S}^{\mathsf{Random}}$ by inserting a CDH challenge in the random points exchanged during some selected sessions.

Suppose some user U successfully completes a session s. Because real keys are revealed both in simulations $\mathcal{S}^{\mathsf{Real}}$ and $\mathcal{S}^{\mathsf{Random}}$ whenever (U, s) has no honest matching session, E must rely on successful sessions where there is at least one honest matching session to distinguish between the two simulations. Because KDF is modeled by a random oracle,

– either E makes at least one KDF-query containing a correct key material for some pair of honest matching sessions (event QH)
– or it has advantage 0 in distinguishing between simulations $\mathcal{S}^{\mathsf{Real}}$ and $\mathcal{S}^{\mathsf{Random}}$.

We therefore only need to bound $\mathsf{P}(\mathsf{QH})$, in simulation $\mathcal{S}^{\mathsf{Real}}$ or $\mathcal{S}^{\mathsf{Random}}$. To this end, transform these simulations as follows: S guesses a session index (U, s) for which QH occurs and U is the session initiator, and introduces a CDH challenge in the random points of U for this session, and of one of its honest matching session $(\mathsf{U'}, \mathsf{s'})$. Because U is the session initiator, it sends its random point first, and any real partner of U receives it before sending its own. Therefore S does not need to guess $(\mathsf{U'}, \mathsf{s'})$ in advance. Key confirmations cannot be properly computed by S; however, random values can be used as placeholders. Once again, because of the random oracle model used for KDF, E does not see the change before it performs the right KDF-query.

To extract a CDH answer, S processes all KDF-queries relevant to the target session, and chooses at random one of the corresponding candidates.

This proof yields a loss factor between $\mathsf{P}(\mathsf{QH})$ and the probability of S to break CDH equal to $q_s q_{\mathsf{KDF}}$, because S guesses a target session and chooses a CDH answer candidate.

It would be tempting to use the CDH random self-reducibility to introduce the challenge in all simulated sessions and eliminate the factor q_s. The problem with this approach is that it does not allow S to simulate correctly sessions between a user U and a user $\mathsf{U'}$ impersonated by E. In such a case, S cannot compute the resulting session key, but E can; in particular, S is not able to decide if a confirmation key output by E is correct or not[2].

6.2 Game G_{ma}

To bound E's success probability in game G_{ma}, we tie it to the probability of the simulator of solving the HCDH problem, which itself reduces to CDH (section 5.)

Introducing the HCDH challenge in the simulation of game G_{ma} requires to guess the first session (U, s) where mutual authentication is defeated, and the intended correspondent $\mathsf{U'}$ of U for this session. The HCDH challenge is then introduced in the

[2] things would be different in a gap group, where S would have access to a DDH oracle.

public keys corresponding to these identities. This translates into the final security bound into a loss factor depending on the number of users simulated and the number of sessions.

The difficulty with this simulation is to deal with sessions involving U or U': since S does not know the private keys of these users, it heavily relies on random oracle programmability to make consistent simulations.

As for game G_{ror}, it seems a better approach would be to try to extract a HCDH answer from *any* session $(\mathsf{U}, \mathsf{s}')$ with intended correspondent U' instead of focusing on (U, s): this would save the term q_s in the final security reduction. There is however an issue with that idea: in "target" sessions where S tries to extract a HCDH answer, S is unable to emit correct key confirmations or to check key confirmations sent by E. Therefore with several target sessions, E could use some of them to test the behavior of the simulator w.r.t. incorrect key confirmations, thereby distinguishing with arbitrary high probability between unmodified game G_{ma} and the game with the HCDH challenge introduced.

Acknowledgements

The authors were supported in part by the European Commission through the IST Program under Contract IST-2002-507932 ECRYPT.

References

1. M. Abdalla, P.-A. Fouque, and D. Pointcheval. Password-Based Authenticated Key Exchange in the Three-Party Setting. In S. Vaudenay, editor, *Public Key Cryptography*, volume 3386 of *LNCS*, pages 65–84. Springer-Verlag, 2005.
2. P. Beguin and J. J. Quisquater. Fast Server-Aided RSA Signatures Secure Against Active Attacks. In *Advances in Cryptology – Crypto'95*, volume 963 of *LNCS*, pages 57–69. Springer-Verlag, 1995.
3. M. Bellare, R. Canetti, and H. Krawczyk. A modular Approach to the design and Analysis of Authentication and Key Exchange Protocols (extended abstract). In *STOC '98*, pages 419–428. ACM Press, 1998.
4. M. Bellare and P. Rogaway. Entity Authentication and Key Distribution. In *Advances in Cryptology – Crypto '93*, volume 773 of *LNCS*, pages 232–249. Springer-Verlag, 1994.
5. R. Canetti and H. Krawczyk. Analysis of Key-Exchange Protocols and Their Use for Building Secure Channels. In *Advances in Cryptology – Eurocrypt'01*, volume 2045 of *LNCS*, pages 453–474, London, UK, 2001. Springer-Verlag.
6. R. Canetti and H. Krawczyk. A Universally Composable Notions of Key Exchange and Secure Channels. In *Advances in Cryptology – Eurocrypt'02*, volume 2332 of *LNCS*, pages 337–351, London, UK, 2002. Springer-Verlag.
7. W. Diffie and M. E. Hellman. New Directions in Cryptography. *IEEE Transactions on Information Theory*, 1976.
8. S. Hohenberger and A. Lysyanskaya. How to Securely Outsource Cryptographic Computations. In *TCC '05*, volume 3378 of *LNCS*. Springer Verlag, 2005.
9. B. S. Kaliski Jr. An Unknown Key-share Attack on the MQV Key Agreement Protocol. *ACM Trans. Inf. Syst. Secur.*, 4(3):275–288, 2001.
10. H. Krawczyk. SIGMA: The 'SIGn-and-MAc' Approach to Authenticated Diffie-Hellman and Its Use in the IKE-Protocols. In D. Boneh, editor, *Proceedings of CRYPTO 2003*, volume 2729 of *LNCS*, pages 400–425. Springer Verlag, 2003.
11. H. Krawczyk. HMQV: A High-Performance Diffie-Hellman Protocol. In Victor Shoup, editor, *Proceedings of CRYPTO 2005*, volume 3621 of *LNCS*, pages 546–566. Springer-Verlag, August 2005.
12. T. Matsumoto, K. Kato, and H. Imai. Speeding Up Secret Computations with Insecure Auxiliary Devices. In *Advances in Cryptology – Crypto' 88*, volume 403 of *LNCS*, pages 497–506. Springer-Verlag, 1988.
13. T. Matsumoto, Y. Takashima, and H. Imai. On Seeking Smart Public-key Distribution Systems. *Transactions of the IECE of Japan*, E69:99–106, 1986.

14. A. Menezes. Another Look at HMQV. Cryptology ePrint archive, Report 2005/205, available at `http://eprint.iacr.org`.
15. A. Menezes, M. Qu, and S. Vanstone. Some New Key Agreement Protocols Providing Mutual Implicit Authentication. *Workshop on Selected Areas in Cryptography (SAC '95)*, pages 22–32, 1995.
16. A. Menezes, P. van Oorschot, and S. Vanstone. *Handbook of Applied Cryptography.* CRC Press, 1996.
17. D. Naccache, D. M'Raïhi, S. Vaudenay, and D. Raphaeli. Can D.S.A. be Improved? Complexity Trade-Offs with the Digital Signature Standard. In *Advances in Cryptology – Eurocrypt'94*, volume 950 of *LNCS*, pages 77–85. Springer-Verlag, 1994.
18. P. Q. Nguyen and J. Stern. The Bguin-Quisquater Server-Aided RSA Protocol from Crypto '95 is not Secure. In *Advances in Cryptology – Asiacrypt'98*, volume 1514 of *LNCS*, pages 372–379. Springer-Verlag, 1998.
19. P. Q. Nguyen and J. Stern. The Two Faces of Lattices in Cryptography. In J. Silverman, editor, *Proc. of Cryptography and Lattices Conference*, volume 2146 of *LNCS*, pages 146 – 180. Springer-Verlag, 2001.
20. O. Chevassut, P.-A. Fouque, P. Gaudry, and D. Pointcheval. Key Derivation and Randomness Extraction. Cryptology ePrint archive, Report 2005/061, available at `http://eprint.iacr.org`.
21. B. Pfitzmann and M. Waidner. Attacks on Protocols for Server-aided RSA Computation. In *Advances in Cryptology – Eurocrypt'92*, volume 658 of *LNCS*, pages 153–162. Springer-Verlag, 1992.
22. D. Pointcheval and J. Stern. Security Arguments for Digital Signatures and Blind Signatures. *Journal of Cryptology*, 13(3):361–396, 2000.
23. V. Shoup and A. Rubin. Session Key Distribution Using Smart Cards. In *Advances In Cryptology - Eurocrypt'96*, volume 1070 of *LNCS*, pages 321–331. Springer Verlag, 1996.
24. Standard for Efficient Cryptography Website. `http://www.secg.org/`.

A Security Proof

A.1 Instantiation of the Model

All simulations are parametrized by a group $(G, +)$ of prime order p and generator P, the number of users simulated n_u, the size h of the output of H, the size k of the output of KDF, the size u of the random value z used by an initiator of a session, the number q_h of H-queries, q_{KDF} of KDF-queries and q_s of $\mathsf{Initiate}$-queries. It is assumed that $2^h < p$.

As seen in figures 4 and 5, an authentication session has two rounds, and the answer to the second round differs in the initiator and in the responder case.

This translates as follows into $\mathsf{t} = \mathsf{IniAuth}(\mathsf{U})$ and $\mathsf{SendAuth}(\mathsf{t}, M)$ queries: the input message M can parse as $1||\mathsf{t}||c$ for the first pass, or $2||\mathsf{t}||\mathsf{ID}_{\mathsf{U}'}||R'$ for the second pass, where $c \in 0, 1$ indicates whether authenticating information for the initiator $(c = 0)$ or the responder $(c = 1)$ should be answered to E, ID_{U} and $\mathsf{ID}_{\mathsf{U}'}$ are valid identities, and $R' \in G$. For the first pass, $\mathsf{SendAuth}$ outputs values in G; for the second pass, $\mathsf{SendAuth}$ outputs values in $\mathbb{Z}_q$ (case $c = 0$) or in $\mathbb{Z}_q \times \{0, 1\}^u$ (case $c = 1$.)

We will use several times Shoup's lemma: if two games A and B are equivalent except if some event of probability P occurs, then there is the following bound on the difference of E's advantages in game A and B: $|\mathsf{Adv}_A - \mathsf{Adv}_B| \leq 2\mathsf{P}$.

A.2 Simulations $\mathcal{S}^{\mathsf{Real}}$ and $\mathcal{S}^{\mathsf{Random}}$

First, the simulator draws n_u random secret keys $s_1, \ldots, s_{n_u}$ in $\mathbb{Z}_p$. All queries are answered in natural way.

Authentication is managed as follows. First, incorrect sequence of queries result in S aborting the simulation: an $\mathsf{SendAuth}(1||\mathsf{t}||c)$ query is invalid if no $\mathsf{IniAuth}(\mathsf{U})$

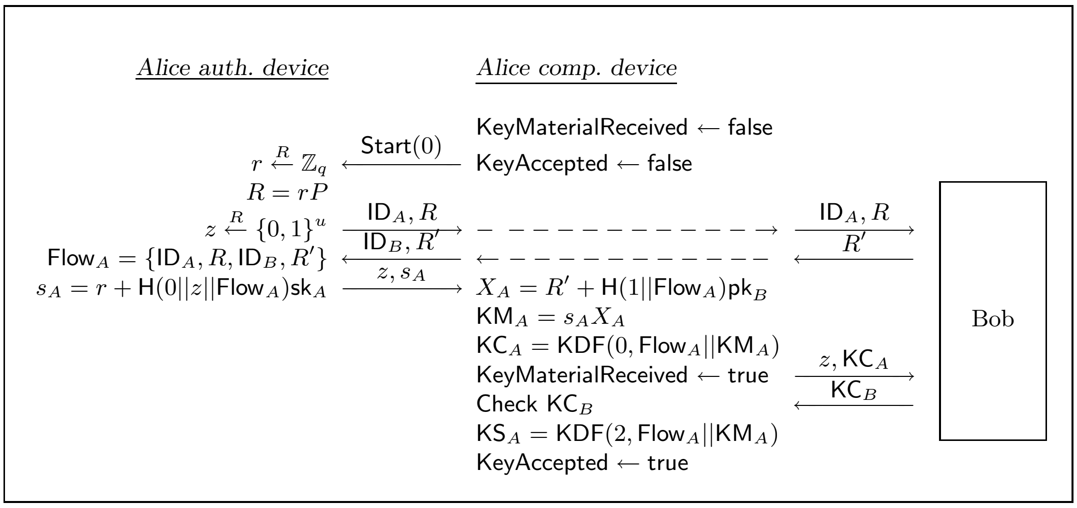
Alice auth. device
Alice comp. device
KeyMaterialReceived ← false
Start(0)
$r \xleftarrow{R} \mathbb{Z}_q$
KeyAccepted ← false
$R = rP$
$z \xleftarrow{R} \{0,1\}^u$
ID_A, R
ID_A, R
ID_B, R'
R'
$\mathsf{Flow}_A = \{\mathsf{ID}_A, R, \mathsf{ID}_B, R'\}$
z, s_A
$s_A = r + \mathsf{H}(0||z||\mathsf{Flow}_A)\mathsf{sk}_A$
$X_A = R' + \mathsf{H}(1||\mathsf{Flow}_A)\mathsf{pk}_B$
$\mathsf{KM}_A = s_A X_A$
$\mathsf{KC}_A = \mathsf{KDF}(0, \mathsf{Flow}_A||\mathsf{KM}_A)$
KeyMaterialReceived ← true
z, KC_A
Check KC_B
KC_B
$\mathsf{KS}_A = \mathsf{KDF}(2, \mathsf{Flow}_A||\mathsf{KM}_A)$
KeyAccepted ← true
Bob

Fig. 4. 4-pass MQV-p honest execution with an authentication token, point of view of Alice

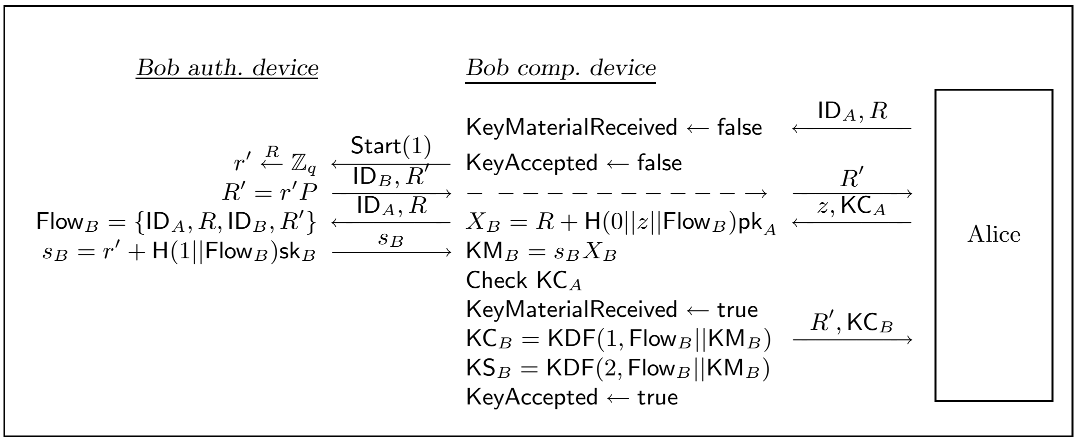
Bob auth. device
Bob comp. device
ID_A, R
KeyMaterialReceived ← false
Start(1)
$r' \xleftarrow{R} \mathbb{Z}_q$
KeyAccepted ← false
ID_B, R'
R'
$R' = r'P$
ID_A, R
z, KC_A
$\mathsf{Flow}_B = \{\mathsf{ID}_A, R, \mathsf{ID}_B, R'\}$
$X_B = R + \mathsf{H}(0||z||\mathsf{Flow}_B)\mathsf{pk}_A$
s_B
$s_B = r' + \mathsf{H}(1||\mathsf{Flow}_B)\mathsf{sk}_B$
$\mathsf{KM}_B = s_B X_B$
Check KC_A
KeyMaterialReceived ← true
R', KC_B
$\mathsf{KC}_B = \mathsf{KDF}(1, \mathsf{Flow}_B||\mathsf{KM}_B)$
$\mathsf{KS}_B = \mathsf{KDF}(2, \mathsf{Flow}_B||\mathsf{KM}_B)$
KeyAccepted ← true
Alice

Fig. 5. 4-pass MQV-p honest execution with an authentication token, point of view of Bob

answered t, an $\mathsf{SendAuth}(2||\mathsf{t}||\mathsf{ID}_{\mathsf{U}'}||R')$ query is invalid if no $\mathsf{SendAuth}(1||\mathsf{t}||c)$ was made, etc.

On query $\mathsf{SendAuth}(1||\mathsf{t}||c)$, where t corresponds to an authentication session initiated for some user U, a random value r is drawn in $\mathbb{Z}_p$, z is drawn in $\{0,1\}^u$ and the association (t, r, z, c) is remembered; rP is answered. On query $\mathsf{SendAuth}(2||\mathsf{t}||\mathsf{ID}_{\mathsf{U}'}||R')$, $(r + \mathsf{H}(0||z||\mathsf{Flow})s_{\mathsf{U}}, z)$ is answered if $c = 0$, and $r + \mathsf{H}(1||\mathsf{Flow})s_{\mathsf{U}}$ is answered if $c = 1$, with $\mathsf{Flow} = \{\mathsf{ID}_{\mathsf{U}}, R, \mathsf{ID}_{\mathsf{U}'}, R'\}$.

The key revealed to E when a session (U, s) ends successfully is the real key in simulation $\mathcal{S}^{\mathsf{Real}}$, and $\mathsf{H}_0(\mathsf{View}(\mathsf{U}, \mathsf{s}))$ where H_0 is a random oracle known only to S in simulation $\mathcal{S}^{\mathsf{Random}}$.

A.3 Game G_{ror}

Here we compute an upper bound for the maximum probability $\mathsf{Adv}_{\mathsf{ror}}$ to distinguish between simulations $\mathcal{S}^{\mathsf{Real}} = \mathcal{S}_0^{\mathsf{Real}}$ and $\mathcal{S}^{\mathsf{Random}} = \mathcal{S}_0^{\mathsf{Random}}$.

As explained in the sketch of proof, we need only to bound the probability for E to make at least one KDF-query containing a correct key material for some pair of honest matching sessions (event QH.)

Define a game G^1_{ror} where collisions on random points sent by S are excluded. Since one random point is sent for each session simulated, the probability p_c that no collision occurs satisfies

$$p_c \geq \frac{p(p-1)\dots(p-q_s)}{p^{q_s}} \geq \left(1-\frac{q_s}{p}\right)^{q_s} \geq 1-\frac{{q_s}^2}{p}.$$

As a consequence, games G_{ror} and G^1_{ror} cannot be distinguished with advantage above $2\frac{{q_s}^2}{p}$.

Define a game G^2_{ror} derived from game G^1_{ror} where S tries to solve a CDH challenge (X, Y). S draws at random a session number $1 \leq i \leq q_s$. Suppose a i^{th} $\mathsf{Initiate}$-query is performed by E, on users U and $\mathsf{U'}$. This means that U is supposed to initiate a session with $\mathsf{U'}$ as responder. Then, in the real simulation $\mathcal{S}_2^{\mathsf{Real}}$ as in the random one $\mathcal{S}_1^{\mathsf{Random}}$, X is sent. Now, we want to introduce Y as the random point of $\mathsf{U'}$ when it responds to U, that is, in any session where $\mathsf{U'}$ receives as first message $\{\mathsf{U}, X\}$. There might be $\ell \leq q_s$ such sessions: we use Diffie-Hellman random self-reducibility and emit random points of the form $Y + r_iP$ in all of these sessions. If QH occurs, we know that $\ell \geq 1$, because there is at least one session $(\mathsf{U'}, \mathsf{s'})$ where $\mathsf{U'}$ receives $\{\mathsf{U}, X\}$. Key confirmations for the sessions where X or Y has been used are replaced by $H_0(\mathsf{Flow})$ for some private random oracle H_0. If QH occurs or if E makes a KDF-query revealing an inconsistency of the key confirmation generation, a KDF-query of the form $\mathsf{KDF}(i||\mathsf{Flow}||K)$ is made, where K is a candidate for the key material of session, and Flow contains the random points exchanged during the session and allows to identify the targeted session. The r_i are pairwise distinct because collisions are ruled out. A unique targeted session is therefore identified; because S knows the private keys of U and $\mathsf{U'}$, and the r_i, K yields a unique CDH answer candidate.

S finally chooses at random one CDH answer candidate and outputs it. Taking into account the scalar multiplications performed, and using Shamir's trick (so that the time to compute a sum of several scalar multiplications is equal to t_{exp}), we see that its success probability satisfies

$$\mathsf{Succ}_{\mathsf{CDH}}(t + (2q_s+1)t_{\mathsf{exp}}, G) \geq \frac{\mathsf{Adv}_{\mathsf{ror}}(t, G, q_{\mathsf{KDF}}, q_s)}{q_{\mathsf{KDF}}\, q_s} - 2\frac{q_s^2}{p}.$$

Therefore,

$$\mathsf{Adv}_{\mathsf{ror}}(t, G, q_{\mathsf{KDF}}, q_s) \leq q_{\mathsf{KDF}}\, q_s \left[\mathsf{Succ}_{\mathsf{CDH}}(t + (2q_s+1)t_{\mathsf{exp}}, G) + \frac{2q_s^2}{p}\right].$$

A.4 Game G_{ma}

Let $\mathsf{MA\text{-}B}$ be the event whose occurring probability $\mathsf{Succ}_{\mathsf{ma}}$ we want to upper bound: some session (U, s) ends successfully ($\mathsf{KeyAccepted} = \mathsf{true}$) while U has no real partner for that session or while its real partner does not match its intended partner $\mathsf{U'}$. It is further required that no $\mathsf{SendAuth}$ is performed on the authentication device of $\mathsf{U'}$ between the beginning of session (U, s) and the moment when $\mathsf{KeyAccepted}$ to true for session (U, s).

We define a game G^1_{ma} where an instance of a HCDH problem is introduced in the first session where the above event occurs.

A.5 Auxiliary game G^1_{ma}

Guess at the beginning of the game an (ordered) pair of users $(\mathsf{U}, \mathsf{U'})$, a bit b and a session index s. A HCDH challenge $(X = xP, Y = yP)$ will be introduced in the guessed session. Our hope is that the first session where $\mathsf{MA\text{-}B}$ occurs is (U, s), that it has intended partner $\mathsf{U'}$, with U as initiator $(b = 0)$, or responder $(b = 1.)$

Assuming $\mathsf{MA\text{-}B}$ occurs at least once, our guess is right with probability above $(2q_s n_u^2)^{-1}$. $X = xP$ and $Y = yP$ are introduced in the public keys of U and $\mathsf{U'}$.

In the next sections, we consider only the case where the guess of S is correct. Sessions are simulated in different ways depending on the user simulated and on its intended correspondent. Remark that the choice of the simulation to use can be made in time since the intended correspondent of an user is defined *before* that user sends any message.

User Simulated	Intended Correspondent	Session Index	Simulation Type
U	$\mathsf{U'}$	s	"Target" session: we will extract the HCDH answer from it
U	Any	$\mathsf{s'} \neq \mathsf{s}$	Sessions simulated thanks to random oracle programmability
$\mathsf{U'}$	Any	Any	
$\mathsf{V} \neq \mathsf{U}, \mathsf{U'}$	$\mathsf{W} \neq \mathsf{U}, \mathsf{U'}$	Any	Sessions simulated as usual (private keys are known by S)

Simulation of the Target Session. (U, s) is simulated as follows: S chooses as in the unmodified simulation $\mathcal{S}^{\mathsf{Real}}$ a random $r \in \mathbb{Z}_p$ that defines the random point $rP = R$ sent by U. A random value $z \in \{0, 1\}^u$ is also chosen if $b = 0$.

Once R is revealed, if $b = 0$, H-queries of the form $\mathsf{H}(1||\{\mathsf{ID_U}, R, \mathsf{ID_{U'}}, R'\})$ are answered by $\mathsf{H}'(R')$. If $b = 1$, H-queries of the form $\mathsf{H}(0||z||\{\mathsf{ID_U}, R, \mathsf{ID_{U'}}, R'\})$ are answered by $\mathsf{H}'(R')$.

This simulation of H could fail if some input of the form above is already bound to some output value of H. This does not occur with probability above q_{H}/p since R is not known by E before it is revealed.

The game is aborted as soon as a key confirmation is received for session (U, s).

Computation of Key Confirmation for the Target Session. S needs to output a key confirmation for session (U, s) if U is the initiator of the session: in that case, it is supposed to produce its key confirmation first. A random value is sent. E's probability to detect this is estimated in a later section.

Event $\mathsf{MA\text{-}B}$ and Key Material Retrieval. Suppose that $\mathsf{MA\text{-}B}$ occurs on the target session (U, s). Then U receives a key confirmation KC for session (U, s) which has no matching session. KC is supposed to be equal to $\mathsf{KDF}(I)$ where I includes the parties identities, the random points exchanged, and the key material KM. Since no session matches (U, s), no value was bound to $\mathsf{KDF}(I)$ by the simulator; therefore if E did not perform the query $\mathsf{KDF}(I)$ itself, $\mathsf{KDF}(I)$ is bound to a random value when KC is received and as a consequence KC is correct with probability only 2^{-k}. Let HQ be the event "E made the query $\mathsf{KDF}(I)$".

$$\begin{aligned}\mathsf{P}[\{\mathsf{KC} = \mathsf{KDF}(I)\}] &= \mathsf{P}[\{\mathsf{KC} = \mathsf{KDF}(I)\} \cap \mathsf{HQ}] + \mathsf{P}[\{\mathsf{KC} = \mathsf{KDF}(I)\} \cap \mathsf{HQ}^{\mathsf{c}}] \\ &= \mathsf{P}[\{\mathsf{KC} = \mathsf{KDF}(I)\} \cap \mathsf{HQ}] + \mathsf{P}[\{\mathsf{KC} = \mathsf{KDF}(I)\}|\mathsf{HQ}^{\mathsf{c}}]\mathsf{P}[\mathsf{HQ}^{\mathsf{c}}] \\ &\leq \mathsf{P}[\{\mathsf{KC} = \mathsf{KDF}(I)\} \cap \mathsf{HQ}] + 2^{-k}\end{aligned}$$

Next, with probability $\geq (1 - 2^{-k})^{q_{\mathsf{KDF}}-1} \geq 1 - q_{\mathsf{KDF}}2^{-k}$, there is no other KDF-query whose output is equal to $\mathsf{KDF}(I)$. Overall, if MA-B occurs, with probability $(\mathsf{P}[\{\mathsf{KC} = \mathsf{KDF}(I)\}] - 2^{-k})(1 - q_{\mathsf{KDF}}2^{-k})$, the correct key material KM can be extracted from the KDF-queries of E.

Informally, the above reasoning shows that when the probability of breaking mutual authentication (event MA-B) is above 2^{-k}, the corresponding key material KM can be extracted from E's KDF-queries. Note that 2^{-k} is the success probability of an attack where a random key confirmation is used: this is why one cannot expect to extract anything when active attacks succeed with probability 2^{-k} or below.

Answering the HCDH Challenge. S answers the HCDH challenge when the guess it made about the first session where MA-B occurs was right. In that case, it receives a random point R' and a correct key confirmation, from which it extracts the key material $\mathsf{KM} = (r + \mathsf{H}(J)x) + (R' + \mathsf{H}'(R')Y)$ for some known r, J and R'. It can therefore output the corresponding HCDH answer $(R', x(R' + \mathsf{H}'(R')Y)$.

Probability to Detect the Malformed Target Session Key Confirmation. E has a non-zero advantage in distinguishing between the random key confirmation produced and a correctly formed one only if it performs a KDF-query containing the correct value of the key material KM for the target session. Call this event KD. Since we could define a game G^2_{ma} equal to G^1_{ma} except that one of the KDF-queries targeting the key confirmation of session (U, s) is chosen at random, and the corresponding HCDH candidate is extracted and answered,

$$\mathsf{P}[\mathsf{KD}] \leq q_{\mathsf{KDF}}\mathsf{Succ}_{\mathsf{HCDH}}(t', G, q_H)$$

where t' is the execution time of S.

Simulation of Other Sessions Involving U or U'. S must be able to simulate perfectly game G^1_{ma} up to session (U, s) without knowing the discrete logs of the public keys of U and U'. Specifically, it must be able to

- produce correct key confirmations for sessions involving U or U',
- answer to SendAuth queries targeted at U or U'.

Corrupt queries on U or U' do not happen before the end of session (U, s), therefore they do not need to be simulated.

The other issues are solved using random oracle programmability. SendAuth queries are simulated as follows: when S receives a $\mathsf{SendAuth}(1||\mathsf{t}||c)$ query, with t opened for U, it draws h, s at random in $[0, 2^{-h}[$, z in $\{0,1\}^u$, answers $R = \mathsf{s}P - \mathsf{h}X$ and remembers the association $(t, c, \mathsf{h}, \mathsf{s})$. On query $\mathsf{SendAuth}(2||\mathsf{t}||\mathsf{ID}_{\mathsf{V}}||R')$, it sets $\mathsf{H}(I) = h$ with $I = 0||z||\{\mathsf{ID}_{\mathsf{U}}, R, \mathsf{ID}_{\mathsf{V}}, R'\}$ if $c = 0$ and $I = 1||\{\mathsf{ID}_{\mathsf{V}}, R', \mathsf{ID}_{\mathsf{U}}, R\}$ if $c = 1$; S then answers (s, z) if $c = 0$ and s if $c = 1$. The same rules apply to U' with X replaced by Y.

The same technique can be used to produce a valid key material that can be used to derive a key confirmation or the session key itself. Therefore when simulating a session different from the target session, with for example user U as initiator, $R = \mathsf{s}P - \mathsf{h}X$ is sent, with s and h random, and as soon as R' is known, $\mathsf{H}(0||z||\{\mathsf{ID}_\mathsf{U}, R, \mathsf{ID}_\mathsf{V}, R'\})$ is bound to h.

The only way this simulation can fail is if some value is already assigned to $\mathsf{H}(I)$.

Failures in Simulation of H. When U (or U') is the responder of a session, S can bind $\mathsf{H}(I)$ to h *before* U sends R'. Since R' is chosen at random, and I includes R', the probability that $\mathsf{H}(I)$ is already assigned some value is below q_H/p.

When U or U' is the initiator of a session, the story is different: R must be sent first, and only when the random point R' of the responder is known, it is possible to perform the binding $\mathsf{H}(I) = h$. However, z is not known to E, therefore it has probability less than $q_\mathsf{H}2^{-u}$ to force a binding of $\mathsf{H}(I)$.

Remarks about z. The role of the secret value z is to allow session simulation in that step of the proof. Alternatively, one could have added a first round where the responder commits its random point, for example by sending $\mathsf{H}_u(R')$ for some u-bit random oracle H_u. E's cheating probability is then the probability to find a second pre-image of $\mathsf{H}_u(R')$, and is still equal to $q_\mathsf{H}2^{-u}$.

The probability of a failure occurring in *any* session is still below $q_\mathsf{H}\max(2^{-u}, 1/p)$, because each of the q_H H-queries can target at most one session.

Putting Everything Together: Bound for $\mathsf{Succ}_{\mathsf{ma}}$. Since simulating a session costs at most two scalar multiplications, the probability $\mathsf{Succ}_{\mathsf{HCDH}}$ that S breaks HCDH satisfies

$$\mathsf{Succ}_{\mathsf{HCDH}}(t', G, q_H) \geq 1/\alpha\, [\mathsf{Succ}_{\mathsf{ma}^1}(t, G, q_H, q_{\mathsf{KDF}}, q_s) - 2^{-k}]$$

where t_{exp} is the time of a scalar multiplication, $t' = t + 2q_s t_{\mathsf{exp}}$, and

$$\alpha = \frac{2\, q_s\, n_u^2}{(1 - q_{\mathsf{KDF}}2^{-k})(1 - q_\mathsf{H}/p)(1 - q_\mathsf{H}\max(2^{-u}, 1/p))}.$$

Since

$$\begin{aligned}\mathsf{Succ}_{\mathsf{ma}^1}(t, G, q_H, q_{\mathsf{KDF}}, q_s) &\geq \mathsf{Succ}_{\mathsf{ma}}(t, G, q_H, q_{\mathsf{KDF}}, q_s) - \mathsf{P}[\mathsf{KD}] \\ &\geq \mathsf{Succ}_{\mathsf{ma}}(t, G, q_H, q_{\mathsf{KDF}}, q_s) - q_{\mathsf{KDF}}\mathsf{Succ}_{\mathsf{HCDH}}(t', G, q_H)\end{aligned}$$

one gets

$$\mathsf{Succ}_{\mathsf{ma}}(t, G, q_\mathsf{H}, q_{\mathsf{KDF}}, q_s) - 2^{-k} \leq (\alpha + q_{\mathsf{KDF}})\mathsf{Succ}_{\mathsf{HCDH}}(t + 2q_s t_{\mathsf{exp}}, G).$$

B HCDH and CDH

In this section, we show that CDH reduces to HCDH using the forking lemma. E is an attacker against HCDH with advantage $\mathsf{Adv}\,(t, q_\mathsf{H})$. We also suppose that E does not perform twice the same H'-query: this can be achieved by making E memorize past answers. Given a CDH challenge $(X = xP, Y)$, we run E with input (X, Y). Assume E succeed. Then, it returns (R, Z) such that

$$Z = x(R + \mathsf{H}(R)Y) = \mathsf{H}'(R) \times xY + xR$$

Using the forking lemma, we are able to rewind E to form a new successful run with new values for some random oracle outputs. E's output (R', Z') satisfies $R' = R$. Then $Z' - Z = (\mathsf{H}'_2 - \mathsf{H}'_1)xY$, with $\mathsf{H}'_2 \neq \mathsf{H}'_1$ the two oracle answers on input R in the two runs. Finally, the answer of the CDH challenge is $(\mathsf{H}'_2(R) - \mathsf{H}'_1(R))^{-1}(Z' - Z)$.

The splitting lemma is as follows [22]:

Lemma 1 (Splitting Lemma) *Let P a probability on a product space $X \times Y$ and $Q \subset X \times Y$.*

Define $$Q' = \left\{ (x, y) \in Q \,\middle|\, \underset{y' \in Y}{P}[(x, y') \in A] \geq P[Q]/2 \right\}$$

Then $$P[Q'|Q] \geq 1/2$$

With probability less than $2/p$, X or Y is equal to 0. In the other cases, $\mathsf{CDH}(X, Y)$ is a generator of G. Then, if R is not among the H'-queries submitted by the attacker, its probability of success is bounded by 2^{-h} because of the term $\mathsf{H}'(R) \times xY$ in the HCDH relation. Overall, with probability $\varepsilon' \geq \mathsf{Adv} - 2/p - 2^{-h}$, the attacker produces a correct output (R, Z) *and* makes the query $\mathsf{H}'(R)$. Now, let Q_i be the event "E produces a correct output (Z, R), the i-th H'-query of E being $\mathsf{H}'(R)$". Let $\varepsilon'_i = \mathsf{P}[Q_i]$. Since E makes a H'-query at most once, the Q_i are disjoint and

$$\sum_{i \leq q_{\mathsf{H}'}} \varepsilon'_i = \varepsilon'.$$

Let us fix i. The whole behavior of the attacker only depends on its random tape and on the oracle answers. Let us split these inputs into the ones occurring before the i^{th} oracle answer ($x \in X$) and the ones after and including that answer ($y \in Y$.) Let us now apply the splitting lemma 1 with $Q = Q_i$. It states that, given $u = (x, y) \in_R Q_i$, with probability $1/2$, we have

$$\mathsf{P}[Q_{i,x}] \geq \varepsilon'_i/2 \quad \text{with} \quad Q_{i,x} = \{(x, y') \in Q_i \mid y' \in Y\}.$$

Therefore, we can perform two executions of E as follows. The first execution is random. With probability greater than ε', it yields a correct answer (R, Z), and $\mathsf{H}'(R)$ is queried on some query of index i. Let x (resp. y) the inputs of E before (resp. after) the i^{th} query. We run again the same execution with inputs x before query i, but y' after query i. With probability $1/2$, inputs x of the attacker before query i are such that $\mathsf{P}[Q_{i,x}] \geq \varepsilon'_i/2$. In that case, with probability $\varepsilon'_i/2$, E produces again a correct output (Z', R') and $\mathsf{H}'(R')$ is queried on query i. Since inputs before query i are equal in both executions, $R = R'$. Finally, except with probability $1 - 2^{-h}$, answers H'_1 and H'_2 for $\mathsf{H}'(R)$ are different in both executions. If all these conditions are met, $\mathsf{CDH}(X, Y) = xY$ can be easily extracted. Overall, the attacker breaks CDH with probability

$$\mathsf{Succ}_{\mathsf{CDH}} \geq \left[\sum_{i \leq q_{\mathsf{H}'}} \left(\frac{\varepsilon'_i}{2} \right)^2 \right] - 2^{-h} \geq \frac{1}{q_{\mathsf{H}'}} \frac{\varepsilon'^2}{4} - 2^{-h}$$

because of the general means inequality. Finally, if E runs in time t the attacker against CDH runs in time $2t$ and succeeds with probability satisfying

$$\mathsf{Succ}_{\mathsf{CDH}}(2t, G) \geq \frac{[\mathsf{Succ}_{\mathsf{HCDH}}(t, G, q_{\mathsf{H}'}) - 2/p - 2^{-h}]^2}{4q_{\mathsf{H}'}} - 2^{-h}.$$