A preliminary version appears in ISC 2009, Lecture Notes in Computer Science, Springer-Verlag, 2009. This version is dated December 18, 2009.

Security Analysis of the PACE Key-Agreement Protocol

Jens Bender[1] Marc Fischlin[2] Dennis Kügler[1]

[1]Bundesamt für Sicherheit in der Informationstechnik (BSI), Germany
[2]Darmstadt University of Technology, Germany

Abstract. We analyze the Password Authenticated Connection Establishment (PACE) protocol for authenticated key agreement, recently proposed by the German Federal Office for Information Security (BSI) for the deployment in machine readable travel documents. We show that the PACE protocol is secure in the real-or-random sense of Abdalla, Fouque and Pointcheval, under a number-theoretic assumption related to the Diffie-Hellman problem and assuming random oracles and ideal ciphers.

1 Introduction

Authenticated key exchange is a fundamental cryptographic protocol in which two parties, usually called the client and the server, establish a secure key. In a password-based key-agreement protocol both parties only share a low-entropy secret, usually drawn at random from a set of size N. Since the security only relies on this short password an adversary can guess the right password with probability at least $1/N$ and then impersonate another party in an execution (in a so-called online dictionary attack). Ideally, this should also be an upper bound on the adversary's success probability, even if the adversary eavesdrops or actively participates in other protocol executions. In particular, the adversary should not be able to deduce the password of any party in an offline dictionary attack by successfully matching password candidates to executions afterwards.

A widely accepted model to capture the above security requirements is the real-or-random security notion of Abdalla, Fouque and Pointcheval [2], a refinement of the model of Bellare, Pointcheval and Rogaway [5]. The original and the refined model have been accepted as a profound approach to capture security of key agreement protocols and several password based protocols for authenticated key exchange (AKE) have been shown secure via this approach [5, 2, 1]. The real-or-random security model says that an adversary, mounting an active attack on several concurrently running instances of the key agreement protocol, cannot distinguish genuine keys from random strings.

The PACE Protocol. Here we investigate the security of the Password Authenticated Connection Establishment (PACE) protocol. This protocol has been specified by the German Federal Office for Information Security (BSI) to secure the communication between a chip contained in a machine readable travel document and a reader (terminal) [6]. The purpose of PACE is to establish a secure channel based on weak passwords like the personal data of the passport holder. The protocol is currently under standardization of ISO/IEC JTC1/SC17/WG3.

The PACE protocol can be roughly divided into four phases (see also Figure 2 on Page 7): In the first phase the chip sends a random nonce s encrypted with the password to the terminal. In the second phase both parties execute an interactive protocol Map2Point, mapping the nonce to a random generator $\widehat{G}$ of a group, e.g., an elliptic curve (the group parameters are provided by the chip and authenticated by a governmental authority). In the third phase the two parties run a Diffie-Hellman (DH) key agreement on the agreed-upon generator $\widehat{G}$ and use the DH key to derive the actual keys for subsequent use. Finally, both parties conclude the execution by sending some authentication data.

PACE is rather a framework allowing different instantiations than a single protocol. Here we focus on the most prominent version based on elliptic curves. Still, we look at different options to implement the Map2Point protocol in which the nonce is thrown to a random generator. A candidate is the DH-based protocol advocated in [6] where both parties generate a DH key H and define $\widehat{G} = sG + H$ for the generator G of the elliptic curve and the nonce s. Another option is to use a coin-flipping protocol instead to generate H jointly and then again letting $\widehat{G} = sG + H$. A third possibility is to hash into the elliptic curve directly. We discuss these options in more detail later.

Security Result for PACE. In this paper we provide a security analysis of the PACE framework. We remark that the purpose of this work here is not to investigate the design choices of the protocol (which are based on implementation aspects and patent issues) but to analyze PACE as a given protocol with respect to security. Some aspects of the protocol are, of course, security-related and in this case we explore them in more detail.

We analyze PACE in the random oracle model and the ideal cipher model (which have recently been shown to be equivalent [9]). These models entail idealized assumptions about the hash function and cipher deployed in the protocol. Namely, it is assumed that the hash function behaves like a random function, and that the cipher acts like a random permutation. We note that neither model may be instantiable in practice [7, 9]. Yet, security shows that, in order to break the scheme, some weaknesses of these primitives must be exploited.

We also introduce a new Diffie-Hellman-like problem, called *g*eneral *PA*ssword-based *C*hosen-*E*lement Diffie-Hellman (gPACE-DH) problem. This problem basically says that it is infeasible for an adversary to derive the final DH key of PACE, even if the adversary impersonates one of the two parties and biases the outcome of the Map2Point subprotocol. It follows that the gPACE-DH problem is connected to the specific choice of the Map2Point step in PACE. For Map2Point protocols guaranteed to produce a random generator we simply speak of the PACE-DH problem.

Our PACE-DH problems resemble the password-based chosen-base (PCDH) problem of Abdalla et al. [4]. Yet, while the PCDH problem is known to be equivalent to the basic

$$\text{gPACE-DH} \quad \overset{\substack{\text{Map2Point}\\ \text{random}\\ \text{output}}}{\leq} \quad \text{PACE-DH} \quad \leq \quad \text{DL}$$

$$\text{gPACE-DH} \quad \overset{\text{generically}}{\equiv} \quad \text{PACE-DH} \quad \overset{\text{generically}}{\equiv} \quad \text{DH} \overset{\text{generically}}{\equiv} \text{DL}$$

Figure 1: Relationships of the (PACE-)DH problems: gPACE-DH denotes the problem related to Map2Point, PACE-DH covers the special case of Map2Point protocols mapping the nonce to a uniformly distributed generator, DH is the regular Diffie-Hellman problem, and DL the discrete logarithm problem; $A \leq B$ denotes that the hardness of problem A implies hardness of the problem B. In the generic group model all problems are hard.

DH problem [4, 1], hardness of the PACE-DH is not known to imply the DH assumption. We nonetheless show that the PACE-DH problem and the gPACE-DH problem are hard in Shoup's generic group model [15] for the choices of Map2Point discussed above. An overview over the relationship of the DH problem to the PACE-DH problems is given in Figure 1.

Assuming the hardness of the gPACE-DH problem we show that PACE is real-or-random secure in the sense of [2], in the random oracle model and ideal cipher model. We also discuss that the protocol provides forward security.

2 Security Model

We analyze the PACE protocol in the real-or-random security model of Abdalla et al. [2] which extends the model of Bellare et al. [5]. Here we provide an overview over the model, for more information and discussion about the choices see [5] and [2].

Attack Model. The model considers a set of honest participants, also called users. Each participant may run several instances of the key agreement protocol, and the j-th instance of a user U is denoted by U_j or (U, j). Each pair of participants shares a secret password π which may be used multiple times to generate session keys. The password π is chosen randomly from a (public) dictionary with N elements.

To obtain a session key the protocol P is executed between two instances of the corresponding users. An instance is called an initiator or client (resp. respondent or server) if it sends the first (resp. second) message in the protocol. For sake of distinctiveness we often denote the client by A and the server by B.

We consider security against active attacks where the adversary's goal is to distinguish between genuine keys, derived in executions between honest parties, and random keys. This corresponds to the so-called real-or-random setting [2], a stronger model than the original find-then-guess model of [5], where the adversary can see several test keys (instead of a single one only).

Each user instance is given as an oracle to which an adversary has access, basically providing the interface of the protocol instance. By assumption, the adversary is in full control of the network, i.e., decides upon message delivery. The adversary can make the following queries to the oracles:

Execute(A, i, B, j). Causes the users A and B to run the protocol for (fresh) instances i and j. The final output is the transcript of a protocol execution. This query simulates a passive attack where the adversary merely eavesdrops the network.

Send(U, i, m). Causes the instance i of user U to proceed with the protocol when having received message m. The output is the message generated by U for m and depends on the state of the instance. This query simulates an active attack of the adversary where the adversary pretends to be the partner instance.

Reveal(U, i). Returns the session key of the input instance. The query is answered only if the session key was generated and the instance has terminated in accepting state. This query models the case when the session key has been leaked. We assume without loss of generality that the adversary never queries about the same instance twice.

Corrupt(U). The adversary obtains the party's long-term key π. This is the so-called *weak-corruption* model. In the *strong-corruption* model the adversary also obtains the state information of all instances of user U. The corrupt queries model a total break of the user and allow to model forward secrecy.

Test(U, i). The oracle test is initialized with a random bit b. Assume the adversary makes a test query about (U, i) during the attack and that the instance has terminated in accepting state, holding a secret key sk. Then the oracle returns sk if $b = 0$ or a random key sk' from the domain of keys if $b = 1$. If the instance has not terminated yet or has not accepted, then the oracle returns $\perp$. This query should determine the adversary's success to tell apart a genuine session key from an independent random key. We assume again without loss of generality that the adversary never queries about the same instance twice.

In addition, since we work in the random oracle and ideal ciper model where oracles providing a random hash function oracle and an encryption/decryption oracle are available, the attacker may also query these oracles.

Partners, Correctness and Freshness. Upon successful termination we assume that an instance U_i outputs a key sk, the session ID sid, and a user ID pid identifying the intended partner (assumed to be empty in PACE for anonymity reasons). We note that the session ID usually contains the entire transcript of the communication but, for efficiency reasons, in PACE it only contains a fraction thereof. We discuss the implications in more detail in Section 3. We say that instances A_i and B_j are *partnered* if both instances have terminated in accepting state with the same output. In this case the instance A_i is called a partner to B_j and vice versa. Any untampered execution between honest users should be partnered and, in particular, the users should end up with the same key (this correctness requirement ensures the minimal functional requirement of a key agreement protocol).

Neglecting forward security for a moment, an instance (U, i) is called *fresh* at the end of the execution if there has been no $\mathsf{Reveal}(U, i)$ query at any point, neither has there been a $\mathsf{Reveal}(B, j)$ query where B_j is a partner to U_i, nor has somebody been corrupted. Else the instance is called *unfresh*. In other words, fresh executions require that the session key has not been leaked (by neither partner) and that no Corrupt-query took place.

To capture forward security we refine the notion of freshness and further demand from a fresh instance (U, i) as before that the session key has not been leaked through a Reveal-query, and that for each $\mathsf{Corrupt}(U)$-query there has been no subsequent $\mathsf{Test}(U, i)$-query involving U, or, if so, then there has been no $\mathsf{Send}(U, i, m)$-query for this instance at any point. In this case we call the instance *fs-fresh*, else *fs-unfresh*. This notion means that it should not help if the adversary corrupts some party after the test query, and that even if corruptions take place before test queries, then executions between honest users are still protected (before or after a Test-query).

AKE SECURITY. The adversary eventually outputs a bit b', trying to predict the bit b of the Test oracle. We say that the adversary wins if $b = b'$ and instances (U, i) in the test queries are fresh (resp. fs-fresh). Ideally, this probability should be close to 1/2, implying that the adversary cannot significantly distinguish random keys from session keys.

To measure the resources of the adversary we denote by

t	the number of steps of the adversary, i.e., its running time, (counting also all the steps required by honest parties)
q_e	the maximal number of initiated executions (bounded by the number of Send- and Execute-queries),
q_h	the number of queries to the hash oracle, and
q_c	the number of queries to the cipher oracle.

We often write $Q = (q_e, q_h, q_c)$ and say that $\mathcal{A}$ is (t, Q)-bounded.

Define now the AKE advantage of an adversary $\mathcal{A}$ for a key agreement protocol P by

$$\mathbf{Adv}_P^{\text{ake}}(\mathcal{A}) := 2 \cdot \text{Prob}[\mathcal{A} \text{ wins}] - 1$$
$$\mathbf{Adv}_P^{\text{ake}}(t, Q) := \max \left\{ \mathbf{Adv}_P^{\text{ake}}(\mathcal{A}) \;\middle|\; \mathcal{A} \text{ is } (t, Q)\text{-bounded} \right\}$$

The forward secure version is defined analogously and denoted by $\mathbf{Adv}_P^{\text{ake-fs}}(t, Q)$.

3 The PACE Protocol

In this section we describe the PACE framework and options for its subprotocol Map2Point.

3.1 The Main Protocol of PACE

We describe the elliptic curve instantiation of the PACE protocol [6]. Roughly, the chip in the PACE protocol first transmits the authenticated group data $\mathcal{G}$ and a nonce s, encrypted with (the hash value of) the password. The receiver can recover this value with the matching password. Then both parties engage in an interactive protocol $\mathsf{Map2Point}(s)$ to map s to a random group element $\widehat{G}$. This generator is subsequently used to run a Diffie-Hellman key agreement to derive a common key K. Once this key is agreed upon, the parties derive the encryption and authentication keys by hashing K appropriately.

In [6] it is for example recommended to use the following protocol $\mathsf{Map2Point}$: Both parties run another DH key agreement protocol (with the chip making the first step), jointly generating a random group element H, but where the secret nonce s does not enter the computation. Then both parties compute the output of this step as $\widehat{G} = sG + H$. We denote this procedure in the following by $\mathsf{DH2Point}$.

We let $\mathcal{H}$ be a hash function, $\mathcal{C}$ be a block cipher, and $\mathcal{M}$ be a MAC. We use $\mathcal{C}(K; s)$ and $\mathcal{C}^{-1}(K; z)$ to denote the encryption and decryption of s and z, respectively, for a secret key K. Let $\mathcal{G} = (a, b, p, q, G, k)$ be the description of an elliptic curve $y^2 = x^3 + ax + b \bmod p$ where $\langle G \rangle$ is a group of prime order q. The chip (A) and terminal (B) share a secret password π from a dictionary with N elements, chosen at random, and use some mapping to generate the secret key K_π for the block cipher from π. Below we let $K_\pi = \mathcal{H}(\pi || 0)$. Note that we implicitly assume that the parties know the right password when engaging in an interaction, e.g., the user may enter the PIN at the reader or the terminal optically scans the machine readable zone of the passport. The ellptic curve version of the PACE protocol is given in Figure 2.

Figure 2: PACE based on DH over elliptic curves (with generic $\mathsf{Map2Point}$ protocol)

A		B
password π		password π
authenticated EC parameters $\mathcal{G} = (a, b, p, q, G, k)$		
$K_\pi = \mathcal{H}(\pi \vert\vert 0)$		$K_\pi = \mathcal{H}(\pi \vert\vert 0)$
choose $s \leftarrow \mathbb{Z}_q$		
$z = \mathcal{C}(K_\pi, s)$		
	$\xrightarrow{\mathcal{G}, z}$	abort if $\mathcal{G}$ incorrect
		$s = \mathcal{C}^{-1}(K_\pi, z)$
	$\mathsf{Map2Point}(s)$	
	$\overrightarrow{}$ $\overleftarrow{}$	
	jointly generate $\widehat{G}$	
choose $y_A \leftarrow \mathbb{Z}_q^*$		choose $y_B \leftarrow \mathbb{Z}_q^*$
$Y_A = y_A \cdot \widehat{G}$		$Y_B = y_B \cdot \widehat{G}$
	$\xleftarrow{Y_B}$	
abort if $Y_B \notin \langle G \rangle \setminus \{0\}$		
	$\xrightarrow{Y_A}$	abort if $Y_A \notin \langle G \rangle \setminus \{0\}$
$K = y_A \cdot Y_B$		$K = y_B \cdot Y_A$
$K_{\text{enc}} = \mathcal{H}(K \vert\vert 1)$		$K_{\text{enc}} = \mathcal{H}(K \vert\vert 1)$
$K_{\text{mac}} = \mathcal{H}(K \vert\vert 2)$		$K_{\text{mac}} = \mathcal{H}(K \vert\vert 2)$
$K'_{\text{mac}} = \mathcal{H}(K \vert\vert 3)$		$K'_{\text{mac}} = \mathcal{H}(K \vert\vert 3)$
$T_A \leftarrow \mathcal{M}(K'_{\text{mac}}, (Y_B, \mathcal{G}))$		$T_B \leftarrow \mathcal{M}(K'_{\text{mac}}, (Y_A, \mathcal{G}))$
	$\xleftarrow{T_B}$	
abort if T_B invalid	$\xrightarrow{T_A}$	abort if T_A invalid
key$= (K_{\text{enc}}, K_{\text{mac}})$		key$= (K_{\text{enc}}, K_{\text{mac}})$
sid $= (Y_A, Y_B, \mathcal{G})$		sid $= (Y_A, Y_B, \mathcal{G})$
pid $= \epsilon$		pid $= \epsilon$

Remarks. Some remarks about the changes compared to the original protocol in [6] and about underlying assumptions are in order.

Session IDs. In the definition of the protocol only the final values (and the group parameters) enter the session ID. This is in order to spare the parties from saving or processing the transcript data in the execution; for a formal treatment of this issue and general solutions see [10]. It follows that the partner definition is "more loose" than the common definition including the whole transcript in sid. With this loose partnering approach here an adversary may now be able to run a man-in-the-middle attack making the honest parties assume they communicate with someone else, even though they hold the same key. Still, the confidentiality of the key is not affected by this.

The final authentication step. The original scheme uses the output key K_{mac} for the MAC computations in the key-agreement protocol, too. This version, however, may not be provable secure in the [5] and [2] model. The reason is that with the Test query the adversary obtains a random or the genuine secret key, including K_{mac}. Then the adversary can possibly test whether this key part K_{mac} together with Y_A or Y_B and matches the transmitted value T_A or T_B. For the general analysis we therefore suggest to derive an ephemeral MAC key K'_{mac} as $K'_{\text{mac}} = \mathcal{H}(K||3)$ and use this key for authentication. A similar strategy is recommended in [5].

Sorting out trivial group elements. The checks for the group elements Y_A, Y_B to be distinct from the neutral element are necessary to prevent trivial attacks in which the adversary (impersonating one of the parties) simply sends $Y_A = 0$ or $Y_B = 0$ in which case it would also hold that $K = 0$. Note that this also requires that Map2Point guarantees $\widehat{G} \neq 0$ with very high probability. See also the discussion in the next section.

3.2 The Map2Point Protocol

In this section we describe possible instantiations for the Map2Point sub routine. An overview is given in Figure 3. We always implicitly assume that both parties check for the right format of received values, e.g., that H is a group element. We take a closer look at the security requirements for Map2Point in Section 4.2.

The Diffie-Hellman Mapping DH2Point. The DH2Point mapping is based on the Diffie-Hellman key agreement. Both parties generate a DH key H (relative to the generator G in $\mathcal{G}$) by exchanging $X_A = x_A G$ and $X_B = x_B G$ and letting $H = x_A x_B G$. The nonce s in then "added" to this DH key via $\widehat{G} = sG + H$. Note that the parties should also check that $H \neq 0$, otherwise the final output would deterministically depend on the nonce s only.

The Coin-flipping Mapping Coin2Point. Also creates $\widehat{G}$ as $\widehat{G} = sG + H$, but both parties use a coin-flipping protocol to generate the random element H. Namely, party A first generates $X_A = x_A G$ and sends a hash value $\mathcal{H}(X_A)$ of X_A, then party B transmits $X_B = x_B G$ and A finally reveals X_A to B (who checks that this value matches the initial hash). Both parties set $H = X_A + X_B$.

Figure 3: Choices for the $\mathsf{Map2Point}$ Protocol

$\mathsf{DH2Point}$(s)

$$
\begin{array}{lcr}
A & & B \\
\text{choose } x_A \leftarrow \mathbb{Z}_q^* & & \text{choose } x_B \leftarrow \mathbb{Z}_q^* \\
X_A = x_A \cdot G & & X_B = x_B \cdot G \\
& \xleftarrow{\quad X_B \quad} & \\
& \xrightarrow{\quad X_A \quad} & \\
\text{abort if } X_B = 0 & & \text{abort if } X_A = 0 \\
H = x_A \cdot X_B & & H = x_B \cdot X_A \\
\widehat{G} = sG + H & & \widehat{G} = sG + H
\end{array}
$$

$\mathsf{Coin2Point}$(s)

$$
\begin{array}{lcr}
A & & B \\
\text{choose } x_A \leftarrow \mathbb{Z}_q^* & & \text{choose } x_B \leftarrow \mathbb{Z}_q^* \\
X_A = x_A \cdot G & & X_B = x_B \cdot G \\
C_A = \mathcal{H}(X_A) & \xrightarrow{\quad C_A \quad} & \\
& \xleftarrow{\quad X_B \quad} & \\
& \xrightarrow{\quad X_A \quad} & \\
& & \text{abort if } C_A \neq \mathcal{H}(X_A) \\
H = X_A + X_B & & H = X_A + X_B \\
\widehat{G} = sG + H & & \widehat{G} = sG + H
\end{array}
$$

$\mathsf{Hash2Point}(s)$
(assumes h2c function onto curve
and pseudorandom function R)

$$
\begin{array}{lcr}
A & & B \\
& & \text{choose nonce } t \\
& \xleftarrow{\quad t \quad} & \\
\widehat{G} = \mathrm{h2c}(R(s,t)) & & \widehat{G} = \mathrm{h2c}(R(s,t))
\end{array}
$$

$\mathsf{Power2Point}(s)$ (with re-encryption)
(for $\mathbb{Z}_p^*$, $p = wq + 1$)

$$
\begin{array}{lcr}
A & & B \\
& & \text{choose key } K' \\
& & s' = \mathcal{C}(K', s) \\
& & \text{where } s = \mathcal{C}^{-1}(K_\pi, z) \\
& \xleftarrow{\quad K' \quad} & \\
s' = \mathcal{C}(K', s) & & \\
\widehat{G} = (s')^w \bmod p & & \widehat{G} = (s')^w \bmod p
\end{array}
$$

The Hash-into-the-Curve Mapping $\mathsf{Hash2Point}$. Assume we have an efficient function $x \mapsto \mathrm{hash2curve}(x)$ allowing to throw the string x to the curve directly. Possible instantiations are given in [14, 13]. Then the two parties can generate the point $\widehat{G}$ as follows. In addition to using the encrypted nonce s, party B sends another nonce t in clear (after having received s), and both parties finally apply a pseudorandom generator $R(s,t)$ to create x. Set $\widehat{G} = \mathrm{hash2curve}(R(s,t))$.

The Power-to-Group Mapping $\mathsf{Power2Point}$. This method only works for groups over $\mathbb{Z}_p^*$. Here the parties use a function $s \mapsto s^w \bmod p$ for $p = wq + 1$ for w with large prime factors only to map s to a sub group element of $\mathbb{Z}_p^*$. The fact that w does not have small factors ensures that the mapping is statistically close to uniform (given that the value s is uniform). One can again combine this mapping with an interactive generation of H, or with the re-encryption technique discussed in the previous case.

4 Security Assumptions

As remarked above we carry out our security analysis assuming an ideal hash function (random oracle model) and an ideal encryption scheme (ideal cipher model). Basically, the first assumption says that $\mathcal{H}$ acts like a random function to which all parties have access. The second property says that for each key K the mapping $\mathcal{C}(K,\cdot)$ is an independent random permutation and one can evaluate both $\mathcal{C}(K,s)$ and $\mathcal{C}^{-1}(K,z)$ for arbitrary values (K,s) and (K,z).

We also require that the message authentication code $\mathcal{M}$ is unforgeable under adaptively chosen-message attacks (see for instance [12]). We denote by $\mathbf{Adv}_{\mathcal{M}}^{\text{forge}}(t,q)$ a (bound on the) value ϵ for which no attacker in time t can output a new message and a valid tag (after having seen at most q MACs for adaptively chosen messages) with probability more than ϵ.

In addition, we need the number-theoretic assumptions discussed below. At the end of this section we discuss security requirements for the $\mathsf{Map2Point}$ protocol.

4.1 Number-Theoretic Assumptions

For passive adversaries, merely eavesdropping the network, security follows from the DH assumption:

Definition 4.1 (DH Problem) *The DH problem is (t,ϵ)-hard if for any adversary $\mathcal{A}$ running in time t the probability that $y_A y_B G \leftarrow \mathcal{A}(\mathcal{G}, G, y_A G, y_B G)$ is at most ϵ (where the probability is over the choice of $\mathcal{G}$, $y_A, y_B \leftarrow \mathbb{Z}_q$ and the internal coin tosses of $\mathcal{A}$).*

We let $\mathbf{Adv}^{\text{DH}}(t)$ denote a (bound on the) value ϵ for which the DH problem is (t,ϵ)-hard.

THE PACE-DH PROBLEM. Active adversaries, injecting messages, can usually contribute to the input to the DH problem and we thus require a stronger assumption based on the *PA*ssword-based *C*hosen-*E*lement (PACE) DH problem. Assume that we are given N values s_i from $\mathbb{Z}_q$ (each value corresponding to $\mathcal{C}^{-1}(K_\pi, z)$ for a possible password π) of which one corresponds to the actual password s_k. Potentially, these values s_i are biased by the adversary through its choice of z so we precautiously let the adversary fully determine them (with the only restriction that they are distinct).

Suppose further that we are given a random group element H (generated via $\mathsf{Map2Point}$ and possibly known to the adversary), as well as $y_B(s_k G + H)$ for a random y_B (for the value Y_B sent by an honest party). Then the adversary's task is to find a group element Y_A (i.e., the Y_A sent in the protocol) and a key K such that $K = y_B Y_A$. Since the adversary may try different possibilities for K, below we let the adversary output a set of q_ℓ possible key values $K_1, \ldots, K_{q_\ell}$.

We first remark that the setting above corresponds to the case that the adversary impersonates the chip. This means that the adversary can adaptively decide upon his choices after seeing the group elements (representing the chip's choices). The case that the adversary plays the terminal is a special case where the adversary first ignores parts of the data. We also remark that, while we consider concurrent executions of the key agreement protocol, for the analysis it suffices to consider our interactive number-theoretic problem in a somewhat isolated setting.

Note that we cannot exclude trivial guessing strategies for our problem. That is, if the adversary manages to guess k it can simply set $Y_A = s_k G + H$ and later choose $K = Y_B$. Similarly, it can choose any linear transformation $Y_A = a(s_k G + H)$ and $K = aY_B$ for $a \in \mathbb{Z}_q$ (also covering the case that $a = 0$ in which case the other party aborts). Hence, there is always an adversarial strategy with success probability at least $1/N$. Yet, this should be close to optimal:

Definition 4.2 (Password-Based Chosen-Element DH Problem) *The password-based chosen-element DH problem is (t, N, q_ℓ, ϵ)-hard if for any adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ running in total time t the probability that the following experiments returns 1 is most $\frac{1}{N} + \epsilon$:*

pick $\mathcal{G}$ (including a generator G)
let $(\mathsf{st}, s_1, \ldots, s_N) \leftarrow \mathcal{A}_0(\mathcal{G}, G, N)$
where $s_1, \ldots, s_N$ are pairwise distinct and st is some local state
pick $H \leftarrow \langle G \rangle$, $y_B \leftarrow \mathbb{Z}_q$ and $k \leftarrow \{1, 2, \ldots, N\}$
let $(Y_A, K_1, \ldots, K_{q_\ell}) \leftarrow \mathcal{A}_1(\mathsf{st}, y_B(s_k G + H), H)$
output 1 iff $Y_A \neq 0$ and $K_i = y_B Y_A$ for some $i \in \{1, 2, \ldots, q_\ell\}$

We let $\mathbf{Adv}^{\text{PACE-DH}}(t, N, q_\ell)$ denote a (bound on the) value ϵ for which the PACE-DH problem is (t, N, q_ℓ, ϵ)-hard.

On the Hardness of the PACE-DH Assumption. The hardness of the PACE-DH problem implies hardness of the discrete logarithm problem (see Section 4.2). We note that the PACE-DH problem resembles the password-based chosen-basis problem of Abdalla et al. [4, 1]. Yet, while that problem has been proven to be equivalent to the DH problem [4, 1] (albeit with a loose security reduction),[1] we are not aware if the PACE-DH problem here is also infeasible assuming the hardness of the DH problem. However, in the generic model of Shoup [15] the problem is also as hard as the DH problem, indicating that only "clever" attacks exploiting the group representation can make a difference in comparison to the regular DH problem. We discuss this in Appendix A.

The gPACE-DH Problem. In the PACE-DH problem above the group element H is assumed to be random. In the actual protocol execution, however, it depends on the execution of protocol $\mathsf{Map2Point}$ in which the adversary may control one of the parties. Hence, in the general PACE-DH problem we mimic the generation of H via $\mathsf{Map2Point}$ and thus lend the adversary more power in generating H:

Definition 4.3 (General Password-Based Chosen-Element DH Problem) *The general password-based chosen-element DH problem is (t, N, q_ℓ, ϵ)-hard (with respect to* $\mathsf{Map2Point}$*)*

[1]Note that the similar chosen-basis *decisional* Diffie-Hellman problems of Abdalla and Pointcheval [3] have been shown to be insecure by Szydlo [16]; Szydlo's attacks do not transfer to the computational counterparts, though.

if for any adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1, \mathcal{A}_2)$ running in total time t the probability that the following experiments returns 1 is most $\frac{1}{N} + \epsilon$:

pick $\mathcal{G}$ (including a generator G)
let $(\mathsf{st}_0, s_1, \ldots, s_N) \leftarrow \mathcal{A}_0(\mathcal{G}, G, N)$
 where $s_1, \ldots, s_N$ are pairwise distinct and st_0 is some local state
pick $y_B \leftarrow \mathbb{Z}_q$ and $k \leftarrow \{1, 2, \ldots, N\}$
let $\widehat{G}$ be the local output of the honest party in an execution of $\mathsf{Map2Point}(s_k)$,
 where $\mathcal{A}_1(\mathsf{st}_0)$ controls the other party (and generates the local state st_1).
let $(Y_A, K_1, \ldots, K_{q_\ell}) \leftarrow \mathcal{A}_2(\mathsf{st}_1, y_B\widehat{G})$
output 1 iff $Y_A \neq 0$ and $K_i = y_B Y_A$ for some $i \in \{1, 2, \ldots, q_\ell\}$

We let $\mathbf{Adv}^{\text{gPACE-DH}}_{\mathsf{Map2Point}}(t, N, q_\ell)$ denote a (bound on the) value ϵ for which the gPACE-DH problem is (t, N, q_ℓ, ϵ)-hard (with respect to $\mathsf{Map2Point}$).

PACE-DH vs. gPACE-DH. Using the coin-flipping protocol $\mathsf{Coin2Point}$ for $\mathsf{Map2Point}$ the output H is (statistically close to) uniformly distributed –and simulatable— and thus security holds under the basic PACE-DH problem. Next consider the $\mathsf{DH2Point}$ protocol which generates H as the DH key from X_A and X_B. Then the hardness of the gPACE-DH problem clearly implies hardness of the PACE-DH problem for $\mathsf{DH2Point}$. That is, given adversary $\mathcal{A}_{\text{PACE}}$ breaking the case of a random H we can easily build an adversary $\mathcal{A}_{\mathsf{DH2Point}}$ against the gPACE problem for $\mathsf{DH2Point}$ by simply following the DH key agreement honestly, such that H is a random element. Then any solution to the random case returned by $\mathcal{A}_{\text{PACE}}$ also gives a solution to the $\mathsf{DH2Point}$ case. The converse is not known to hold, essentially because the DH key agreement may not yield a uniformly distributed element (if the honest party goes first). Still, we note again that in the generic group model both problems are hard, and we indicate in Appendix A that both problems may indeed be close even in the non-generic case. Some potential advantages of the $\mathsf{DH2Point}$ approach over $\mathsf{Coin2Point}$ are discussed in the next section.

4.2 Requirements for the Map2Point Protocol

In this section we discuss security aspects of the $\mathsf{Map2Point}$ protocol. We subsume the choices for $\mathsf{Map2Point}$ under some general form.

Canonical Map2Point Protocols. To define the security requirements for $\mathsf{Map2Point}$ in the PACE protocol formally we need a special property basically saying that $\mathsf{Map2Point}$ consists of an interactive step $\mathsf{RndPoint}()$ where both parties jointly generate some randomness, say, a random group element H or a key K'. In the sequel we denote this randomness simply by H. This step should be independent of the nonce s and only depend on the public data (including transmitted values). Only in the final local step the parties compute $\widehat{G}$ from this value H and the nonce s via a non-interactive algorithm $\mathsf{NncPoint}(s, H)$.

We call a protocol $\mathsf{Map2Point}$ *canonical* if, in addition to the aforementioned structure, the protocol is non-trivial in the sense that for any $s \in \mathbb{Z}_q$ the output $\widehat{G}$ of an execution of

$\mathsf{Map2Point}(s)$ *between honest parties* satisifies $\widehat{G} \neq 0$. Note that for $\mathsf{DH2Point}$ and $\mathsf{Coin2Point}$, for example, there is a small probability of $1/q$ that the output of $\mathsf{Map2Point}$ is 0, namely, if $H = -sG$. We ignore this small term to simplify the presentation and merely note that such cases can be easily thwarted by testing for trivial values and setting $\widehat{G} = G$ in this case.

NECESSARY SECURITY REQUIREMENTS. To be suitable for the gPACE-DH problem any canonical $\mathsf{Map2Point}$ protocol must satisfy the following properties:

Proposition 4.4 *For a canonical protocol* $\mathsf{Map2Point}$ *it can only hold* $\boldsymbol{Adv}^{gPACE\text{-}DH}_{\mathsf{Map2Point}}(t, N.q_\ell) \leq \epsilon$ *if the following is true:*

Collision-Resistance. Consider distinct, adversarial chosen values $s_1, \ldots, s_N \in \mathbb{Z}_q$ *and an execution of* $\mathsf{RndPoint}()$ *where the adversary (running in time at most* $t - \Theta(N \cdot \mathit{Time}(\mathsf{NncPoint})))$ *may control one of the parties, and where the honest player and the adversary output* H*. Then we have* $\mathsf{NncPoint}(H, s_i) = \mathsf{NncPoint}(H, s_j)$ *for* $i \neq j$ *with probability at most* $\epsilon' = N\epsilon$*.*

Hardness of Discrete-Logarithms. Consider distinct, adversarial chosen values $s_1, \ldots, s_N \in \mathbb{Z}_q$ *and an execution of* $\mathsf{RndPoint}()$ *where the adversary (running in time at most* $t - \Theta(N \cdot \mathit{Time}(\mathsf{NncPoint}) + N \cdot \mathit{Time}(\mathit{Exp}_{\mathcal{G}})))$ *may control one of the parties, and where the honest player and the adversary output* H*. Then the probability that the adversary also outputs a value* $\log_{\mathsf{NncPoint}(H,s_i)} \mathsf{NncPoint}(H, s_j)$ *for some* $i \neq j$ *is at most* $\epsilon' \leq N\epsilon$*. (In particular, for* $\mathsf{Map2Point} = \mathsf{DH2Point}$ *the probability for outputing* $\log_G H$ *must be at most* $N\epsilon$*.)*

The second property indicates that deterministic protocols $\mathsf{Map2Point}$ must be treated with special care because then the outcome of the protocol may be under full control of the adversary. It must then be ensured that the adversary cannot find $s_1, \ldots, s_N$ such that it knows the discrete logarithm of $\mathsf{Map2Point}(s_j)$ with respect to $\mathsf{Map2Point}(s_i)$ for some $i \neq j$. One remedy against such an attack may then be a strong cipher for which the adversary cannot find such s_i, s_j which map to the same ciphertext (for the corresponding hashed passwords). While this is true in the ideal cipher model it may be much harder to achieve this property for concrete instantiations.

Proof (of Proposition 4.4). It suffices to show the second case, because the for a collision we obviously have $\log_{\mathsf{NncPoint}(H,s_i)} \mathsf{NncPoint}(H, s_j) = 1$.

Assume that there is an adversary $\mathcal{B}$ (with the given running time) finding the discrete logarithm with probability more than ϵ'. Then we construct a successful adversary $\mathcal{A}$ in the gPACE-DH game. Adversary $\mathcal{A}$ initially runs $\mathcal{B}$ to generate $s_1, \ldots, s_N$ and relays the communication from the the $\mathsf{RndPoint}$ execution and its $\mathsf{Map2Point}$ run. Algorithm $\mathcal{A}$ then, upon receiving H and a value a from $\mathcal{B}$, as well as $Y_B = y_B\widehat{G}$ in the game, checks if there are $i \neq j$ with $a = \log_{\mathsf{NncPoint}(H,s_i)} \mathsf{NncPoint}(H, s_j)$. If not, it sets $j = 1$. Adversary $\mathcal{A}$ eventually outputs $Y_A = \mathsf{NncPoint}(H, s_j)$ and $K_1 = Y_B$ as well as $K_2 = aY_B$.

For the analysis note that, if $\mathcal{B}$ succeeds in finding a discrete logarithm, then $\mathcal{A}$ wins with probability at least $2/N$. This is true because the “right” index k is picked at random, independently of the outcome of $\mathsf{RndPoint}$, and thus hits the bad indices with probability at least $2/N$ (such that Y_B or aY_B is a valid solution for j resp. i). Given that a is not a discrete logarithm we again have $k = j = 1$ with probability $1/N$. Overall, adversary $\mathcal{A}$ wins with probability $\frac{1}{N} + \frac{1}{N}\epsilon' > \frac{1}{N} + \epsilon$. Noting that $\mathcal{A}$ runs in time t we obtain a contradiction to the hardness of the gPACE-DH problem. □

SPECIFIC Map2Point CHOICES. Here we consider the security of the previously defined choices for Map2Point:

The DH2Point *Protocol.* Recall that both parties compute $\widehat{G}$ as $\widehat{G} = sG + H$ where H is a Diffie-Hellman key. If both parties in this step are honest then the outcome is clearly a random element H and breaking such an instance would require to solve the PACE-DH problem (instead of the gPACE-DH problem). If the adversary impersonates the terminal and sends X_B first, then, unless $X_B = 0$, the value H is random as well (for $X_B = 0$ the honest party aborts). It is unclear how the adversary can take advantage (in breaking the key agreement protocol) if it chooses X_A after receiving X_B from an honest party, without knowing the discrete logarithm of X_A to G. But assuming that the adversary knows the discrete log, the value H is a random group element as in the PACE-DH problem (or the adversary is able to deduce the discrete logarithm of X_B to G and the discrete logarithm problem is thus easy).

The Coin2Point *Protocol.* In the coin-flipping case where A first commits to X_A and then both parties exchange X_A and X_B we set $\widehat{G} = sG + X_A + X_B$. Assuming that the hash function $\mathcal{H}$ acts like a random oracle we obtain a secure solution under the basic PACE-DH problem (as $\mathcal{H}$ allows to decommit arbirtrarily for honest A, and to extract the value X_A from a commitment of a malicious A).

A potential disadvantage compared to DH2Point is that an outsider here can determine $H = X_A + X_B$ from the interaction X_A, X_B between two honest parties, and can thus possible find password candidates. In the DH2Point case the value H is essentially an unknown random element to an eavesdropper (under the decisional Diffie-Hellman assumption) and therefore also "shields" the password. Another point is that the coin-flipping protocol relies on the (perfect) one-wayness of the hash function (which may be a different requirement than for a good key derivation function); any leakage about X_A from $\mathcal{H}(X_A)$ may allow the adversary to bias the outcome H via X_B.

The Hash2Point *Protocol.* Requires that the pseudorandom mapping R generates a (pseudo)-random element, even if the input is partly under control of the adversary. Furthermore, the hashing into the curve must be such that neither side knows the discrete logarithm of the group element.

The Power2Point *Protocol.* The protocol clearly require that the mapping from s' to the group does not allow to compute discrete logarithms easily. This must particularly hold since the re-encryption technique potentially allows a malicious party B to find suitable keys K' after learning s. In the ideal cipher model the adversary, however, has very limited control over the outcome s'.

5 AKE-Security of PACE

We analyze the PACE protocol with respect to general $\mathsf{Map2Point}$ protocols:

Theorem 5.1 *Let* $\mathsf{Map2Point}$ *be canonical and assume that the password is chosen from a dictionary of size* N*. In the random oracle model and the ideal cipher model we have*

$$\begin{aligned}\boldsymbol{Adv}^{ake}_{PACE}(t, Q) \quad \leq \quad & \frac{q_e}{N} + q_e \cdot \boldsymbol{Adv}^{gPACE\text{-}DH}_{\mathsf{Map2Point}}(t^*, N, q_h) \\ & +q_e \cdot \boldsymbol{Adv}^{forge}_{\mathcal{M}}(t^*, 2q_e) + \frac{2q_e N^2 + 8q_e^2 N + q_c q_e}{\min\{q, |Range(\mathcal{H})|\}}\end{aligned}$$

where $t^* = t + O(kq_e^2 + kq_h^2 + kq_c^2 + k^2)$ *and* $Q = (q_e, q_c, q_h)$.

We remark that the time t^* covers the additional time to maintain lists and perform look-ups.

Proof. Correctness of the protocol follows from the correctness of the MAC algorithm $\mathcal{M}$ and the fact that $\mathsf{Map2Point}$ does not return a trivial group element $\widehat{G} = 0$.

We show security via the common game based approach, gradually changing the original attack Game_0 (with random test bit b) via experiments $\mathsf{Game}_1, \mathsf{Game}_2, \ldots$ to a game where the adversary's success probability to predict b is bounded by the guessing probability of $\frac{1}{2}$. Each transition from Game_i to Game_{i+1} will only change the adversary's probability only slightly (depending on cryptographic assumptions), thus showing that the success probability in the original attack cannot be significantly larger than $\frac{1}{2}$. (Formally we can condition on all "bad" events ruled out in the previous games to not happen.)

Technically, we would like to conclude that the adversary never makes a hash query about a Diffie-Hellman key from which an honest party has derived the output keys. If such a query does not occur then, because we deploy a random oracle, the final keys still look random. We show that this is essentially true under the hardness of the gPACE-DH problem (and in the course take advantage of the random oracle and ideal cipher model).

But we also need to take into account attacks where the adversary manages to find *unpartnered* instances but which derive the same keys. In this case the adversary could easily distinguish the answer of a Test-query by posting a Reveal-query for the unpartnered instance (if the instances are partnered then such a Reveal-query is not admissible for a success). We prove that this is guaranteed by the unforgeability of the MAC.

We also remark that we assume that no $\mathsf{Corrupt}$-query takes place in this setting (or else the adversary cannot win). We cover forward security and $\mathsf{Corrupt}$-queries in Section 6. We next define the games.

Description of Game_0. Corresponds to the original attack on the protocol.

Description of Game_1. As Game_0 but abort in case of K_π collisions.

We abort the experiment (declaring the adversary to lose) whenever there are distinct passwords $\pi \neq \pi^*$ yielding the same hash value $K_\pi = \mathcal{H}(\pi||0) = \mathcal{H}(\pi^*||0)$. Since there are at most $\frac{1}{2}N^2$ admissible password pairs in total and $\mathcal{H}$ is a random oracle, the adversary's success probability decreases by at most $\frac{1}{2}N^2/|\text{Range}(\mathcal{H})|$ by the birthday bound. Letting Game_i also denote the event that the adversary successfully predicts the test bit b in Game_i we thus have

$$\text{Prob}[\,\mathsf{Game}_0] \leq \text{Prob}[\,\mathsf{Game}_1] + \frac{N^2}{2 \cdot |\text{Range}(\mathcal{H})|}.$$

Description of Game_2. As Game_1 but abort in case of collisions among decrypted values.

We abort (again declaring the adversary to lose) if there appears some value z in an execution such that for some admissible passwords $\pi \neq \pi^*$ we have $\mathcal{C}^{-1}(K_\pi, z) = \mathcal{C}^{-1}(K_\pi^*, z)$. Since $\pi \neq \pi^*$ implies $K_\pi \neq K_\pi^*$ by the first game and the cipher is ideal, the probability that for any of the at most q_e values z in the executions we have a collision is at most $\frac{1}{2}q_e N^2/q$.

$$\text{Prob}[\,\mathsf{Game}_1] \leq \text{Prob}[\,\mathsf{Game}_2] + \frac{q_e N^2}{2q}.$$

Description of Game_3. As Game_2 but abort in case two keys $K \neq K^*$ of two accepting user instances yield an identical key K_{enc}, K_{mac} or K'_{mac}.

Since there are at most $\frac{1}{2}(2q_e)^2$ of such user instances and the probability that two fixed ones yield a hash collision for one of the output keys is at most $3/|\text{Range}(\mathcal{H})|$, the adversary's success probability only drops by the term $6q_e^2/|\text{Range}(\mathcal{H})|$.

$$\text{Prob}[\,\mathsf{Game}_2] \leq \text{Prob}[\,\mathsf{Game}_3] + \frac{6q_e^2}{|\text{Range}(\mathcal{H})|}.$$

Description of Game_4. As Game_3 but simulate the ideal cipher.

We replace the actual ideal cipher $\mathcal{C}$ by a lazy-sampling like technique. Namely, for honest users we maintain an intially empty list of tuples (A, B, s, z). For each honest party (involved in a protocol instance between A and B) calling $\mathcal{C}$ about (K_π, s) we check the list for an entry (A, B, s, z) and, if there exists one, we return z. Else we pick a random element z, return it and store (A, B, s, z) in the list. For each call of an honest party (involved in instance (A, B)) to $\mathcal{C}^{-1}$ about (K_π, z) we also search for an entry (A, B, s, z) and return s if we find such an entry; else we pick a random s, store (A, B, s, z) and return s.

For the adversary we keep a separate list. For any call of the adversary to $\mathcal{C}$ about (K_π, s) we check if there is already an entry (K_π, s, z) and return z if so; else we return a random value z and store (K_π, s, z). For each call of the adversary to $\mathcal{C}^{-1}(K_\pi, z)$ we search for an entry (K_π, s, z) in the list and return s if such an entry exist, else we pick a random s and return s and store (K_π, s, z).

Note that the two lists may cause inconsistencies between the answers to honest users and to the adversary. However, conditioning on the adversary never making a hash query about a DH key derived by an accepting user instance the execution of Game_3 does not reveal any information about the s-values chosen by honest parties. More formally, let THQ be the event that the adversary makes a so-called *target hash query* about $K||n$ for some $n \in \{1, 2, 3\}$, where K is the key some honest user instance has derived (before computing the MACs).

We next bound the probability for event THQ by describing another game in which we abort if this happens (and then show that under the gPACE-DH assumption this cannot happen too often). To be precise we actually consider the event THQ in Game_3 where it occured for the first time. But since we are only interested in the first target hash query and up to the point where this target hash query is made the modifications from Game_3 to Game_4 cannot affect the adversaries success probability significantly (as shown above), it suffices to consider event THQ in Game_4. An important observation here is that up to the first target hash query the data in Game_4 is independently distributed from the actual passwords of users (because neither the simulated cipher nor the MAC computations reveal anything about the password, the interactive runs of protocol $\mathsf{RndPoint}$ are also password-independent, and the group elements in the final DH exchange are distributed independently of $\widehat{G}$).

Conditioning on $\neg\mathsf{THQ}$, the probability of making an accidental query to $\mathcal{C}$ about an s-value chosen by an honest party is at most $q_c q_e/q$. Analogously, if no target hash queries occur, then answering calls of the adversary to $\mathcal{C}^{-1}$ as described above, does not lead to any difference in the success probability.

$$\mathrm{Prob}[\,\mathsf{Game}_3] \leq \mathrm{Prob}[\,\mathsf{THQ}] + \mathrm{Prob}[\,\mathsf{Game}_4 \mid \neg\mathsf{THQ}\,] + \frac{q_c q_e}{q}.$$

Description of Game_5. As Game_4 but stop if the adversary makes a hash query about a DH-key of an accepting user instance (U,i).

We even declare the adversary victorious if it ever submits a query $K||n$ for $n \in \{1,2,3\}$ to the hash oracle $\mathcal{H}$ such that a user instance (U,i) has computed this key and sent out the final MAC (i.e., we even consider instances in which the user may not accept eventually). We claim that this cannot occur with probability more than q_e/N, plus the advantage of breaking the gPACE-DH problem (times q_e). Consider a user instance (U,i) in accepting state and the corresponding execution in which the DH-key K is derived.

We now break the gPACE-DH problem as follows. We are given $(\mathcal{G},G,N)$ as input. We initially make a guess for the execution number between 1 and q_e for which the adversary makes the first test query and, at the same time, a target hash query. Then we simulate Game_5. We wait to receive z in this execution and then output the (possibly then chosen) values $s_1,\ldots,s_N$ for all passwords π and all (unique) derived keys K_π and for each call by the adversary to $\mathcal{C}^{-1}$.

In the predicted execution we run the $\mathsf{Map2Point}$ algorithm with the adversary to obtain $\widehat{G}$ (relaying the communication in the execution and the external $\mathsf{Map2Point}$ instance in the gPACE-DH problem). We then receive $y_B\widehat{G}$ as additional input and feed these data into the execution. We finally pick random keys $K_{\mathrm{enc}}, K_{\mathrm{mac}}, K'_{\mathrm{mac}}$ (instead of querying $\mathcal{H}$) and complete the protocol with the help of these data. When the adversary eventually stops we output Y_A, transmitted in the predicted execution by the adversary or the honest party, and the list $K_1,\ldots,K_{q_h}$ of values appearing in the at most q_h hash queries of the form $K||n$ for $n \in \{1,2,3\}$. (If the adversary impersonates the chip then we output the value Y_B, of course.)

It remains to analyze the probability that we obtain an admissible solution to the gPACE-DH problem. Recall that we fail to win if we output $Y_A = 0$. But this case leads the honest party to abort immediately. Hence, we can assume $Y_A \neq 0$. Also note that all possible nonces $s_1, \ldots, s_N$ for the different passwords are distinct by Game_2 and thus comply with the requirement for the gPACE-DH game. Hence, up to the target hash query the distribution of the data is independent of the password of the user instance, and we make the right execution prediction with probability $1/q_e$, in which case we obtain a valid solution to the gPACE-DH problem whenever the adversary makes a target hash query.

Overall, the success probability cannot decrease by more than

$$\mathrm{Prob}[\,\mathsf{THQ}] \leq \frac{q_e}{N} + q_e \cdot \left(\mathbf{Adv}^{\text{gPACE-DH}}_{\mathsf{Map2Point}}(t^*, N, q_h)\right)$$

From now on we can condition on the adversary not making a hash query about the DH-key of an accepting user instance.

Description of Game_6. As Game_5 but replace keys $K_{\text{enc}}, K_{\text{mac}}$ in Test-queries by random keys.

Note that, since we assume that the adversary never makes a hash query about a DH-key of an accepting user instance, this simulation is perfect unless there is an accepting instance (U, i) having the same DH-key as another instance (U^*, j) but such that the two instances are not partnered. In this case the adversary could make a Reveal-query to party (U^*, j) and could notice the difference to the $\mathsf{Test}(U, i)$ query. Note that Reveal-queries to partnered instances do not lead to a win for the adversary.

Consider a DH-key K derived by some honest party U in an execution i. In particular, (U, i) has accepted and returned keys $(K_{\text{enc}}, K_{\text{mac}})$, session ID $(Y_A, Y_B, \mathcal{G})$ and pid $= \epsilon$. We show that, except with negligible probability, there cannot exist some user U^* in execution j such that this user also accepted with output $(K_{\text{enc}}, K_{\text{mac}})$, sid $= (Y_A^*, Y_B^*, \mathcal{G}^*)$ and pid $= \epsilon$, but such that the two instances are not partnered. According to Game_3 the match of the keys in the output also implies that the derived DH-key K^* of this user instance must match K.

Assume now that (U^*, j) is *not* partnered with (U, i). The only possibility that instance (U^*, j) is not partnered with (U, i) stems from different session IDs, i.e., $(Y_A^*, Y_B^*, \mathcal{G}^*) \neq (Y_A, Y_B, \mathcal{G})$. Note that, in order to make (U^*, j) accept there must be valid MACs computed over the data. If either of the values $(Y_A, \mathcal{G}), (Y_B, \mathcal{G}), (Y_A^*, \mathcal{G}^*), (Y_B^*, \mathcal{G}^*)$ has not been sent by an honest party (with a subsequent MAC) then the adversary must be able to forge MACs.

To be more precise, assume that the adversary finds a valid MAC for a new value. Then we show how to forge a MAC of a new message, given oracle access to a MAC algorithm $\mathcal{M}(K'_{\text{mac}}, \cdot)$ for a random (and unknown) key K'_{mac}. Initially we guess a number between 1 and q_e for an execution. We run the Game_6 controlling the random oracle and with full knowledge of all secret values with one exception: instead of calling $\mathcal{H}$ about $(K||3)$ for the chosen execution we omit this step and use the external oracle to derive the MAC (and subsequently for any other of the at most $2q_e$ MAC computations based on the same DH-key). When the adversary sends a valid MAC in an execution with the same DH-key for a new value then we copy these data and stop.

Note that we guess the first execution in which the DH-key in question appears with probability $1/q_e$. Given a correct guess we find a valid forgery for the MAC with the same probability as the adversary succeeds in the experiment. Note that the data transmitted by the adversary have not been sent by an honest user and have thus never been submitted to MAC oracle.

Hence, the adversary can only inject values $(Y_A, \mathcal{G})$, $(Y_B, \mathcal{G})$,$(Y_A^*, \mathcal{G}^*)$, $(Y_B^*, \mathcal{G}^*)$ with valid MACs generated by honest users in other executions. But once the key K is determined by one instance, in the other instance the honest user contributes a random exponent, say, y_B (the case that the honest user goes second and picks y_A is even simpler). But then the adversary can only choose among the at most $2q_e$ other values chosen by honest parties and send one of them. The probability that any of these at most $2q_e$ values yields, together with y_B the key K on the user's side, is at most $2q_e/q$. Hence,

$$\mathrm{Prob}[\,\mathsf{Game}_5 \mid \neg\mathsf{THQ}\,] \leq \mathrm{Prob}[\,\mathsf{Game}_6] + \frac{2q_e}{q} + q_e \cdot \mathbf{Adv}_{\mathcal{M}}^{\mathrm{forge}}(t^*, 2q_e).$$

Putting the Results Together. In the final game all the keys returned in Test-queries are independent random values and the adversary cannot distinguish them from genuine ones. The success probability is therefore bounded from above by $\frac{1}{2}$ in this game. Hence, the adversary's success probability for the original game cannot be larger than $\frac{1}{2}$ plus any "losses" on the transition to the final game. □

6 Discussion

On Forward Secrecy. The above theorem remains true in the forward-secrecy setting (assuming weak corruptions). Recall that here the adversary may corrupt players and learn the long-lived secret but, if it does, then it can only issue further Test-queries for instances in which both users have been honest. The latter guarantees that, even for insecure long-lived passwords, future executions between honest parties are still protected.

To show forward security we need a slight variant of the gPACE-DH problem in which the adversary first outputs Y_A, then learns k and finally outputs q_ℓ potential keys $K_1, \ldots, K_{q_\ell}$. This (adaptive) version of gPACE-DH for parameters (t, N, q_ℓ) can be shown to be as hard as the (non-adaptive) gPACE-DH problem for parameters (Nt, N, Nq_ℓ). For this simply let the non-adaptive adversary simulate the adaptive adversary up to the point where it outputs Y_A. Instead of outputting Y_A the non-adaptive algorithm internally completes N runs of the adaptive adversary for all N possible choices of k, yielding at most N times q_ℓ possible keys. The non-adaptive adversary finally outputs Y_A and this list of keys, and wins with the same probability as the adaptive adversary.

Forward secrecy of the protocol follows under the adaptive gPACE-DH problem. If a party gets corrupted after a Test-query (in which case an honestly or maliciously chosen Y_A has already been determined, before the adversary learns the password and thus k) computing the DH key would require to solve the adaptive gPACE-DH problem and thus the gPACE-DH problem. If a party gets corrupted before a Test-query then this execution does not involve Send-commands and the data are thus chosen honestly. In particular, one can think of Y_A as being chosen at random before the adversary learns the password. Security then also follows from the adaptive gPace-DH problem.

Untraceability, Unlinkability and Non-Transferability. Here we outline to what extend the PACE protocol obeys other desirable security features. Untraceability means that one cannot trace the chip's identity afterwards. Unlinkability says that transcripts cannot be linked. Non-transferability guarantees that the terminal cannot prove transactions to third parties.

Untraceability holds in a restricted sense. Since different countries will have distinct elliptic curve parameters (especially if they contain a country's signature for authentication) one can at least link transcripts to countries. Other than that, if one eavesdrop an execution between two honest parties, then the chip's identity remains hidden. A similar argument holds for unlinkability. Non-transferability follows from the fact that the terminal could generate the data in a communication itself with the help of the password (assuming that the authenticated elliptic curve data of countries are publicly available anyway).

Random Oracles and Ideal Ciphers. Our analysis is carried out using the combination of two idealized assumptions, namely, the random oracle model and the ideal cipher model. Both models have recently shown to be equivalent [9], and haven been used before to analyze the security of other key agreement protocols (e.g., [5]). The security proof here therefore relies on the assumption that both the hash function and the cipher have been designed well.

Composability. The protocol has been analyzed in the security model of Abdalla et al. [2] and Bellare et al. [5]. Recently, Canetti's universal composition (UC) model has been used to establish security requirements for password-based key agreement [8]. If proven secure in this model then the protocol can be securely composed (with itself and with other protocols) and it can be shown that the derived keys can safely be used to establish secure channels. (We note that in case of machine readable travel documents composability with other protocols may be a minor issue.) Still, we note that the protocol is secure under concurrent executions of parties.

Unfortunately, UC-secure password-based key agreement in the standard model is impossible [8]. This does not rule out that the PACE protocol cannot be proven secure in this setting if one again assumes random oracles or ideal ciphers or authenticated group data. Yet, some (more or less formal) modification of the protocol is still necessary: the UC model requires session identifiers which are known in advance, whereas in the model of [2, 5] the session id is simply set afterwards to be (parts of) the communication transcript.

Acknowledgments

We thank the anonymous reviewers of ISC 2009 for valuable comments. We also thank the participants of the WG 16 sub group for the stimulating discussions, and Rainer Urian for feedback.

References

[1] Michel Abdalla, Emmanuel Bresson, Olivier Chevassut, Bodo Möller and David Pointcheval: *Provably Secure Password-Based Authentication in TLS*, ASIACCS '06, pp. 35–45, ACM Press, 2006.

[2] Michel Abdalla, Pierre-Alain Fouque, and David Pointcheval: *Password-based authenticated key exchange in the three-party setting*, PKC 2005, Lecture Notes in Computer Science 3386, pp. 65–84. Springer-Verlag, 2005.

[3] Michel Abdalla and David Pointcheval: *Interactive Diffie-Hellman Assumptions with Applications to Password-based Authentication*, Financial Cryptography 2005, Lecture Notes in Computer Science 3570, pp. 341–356, Springer-Verlag, 2005.

[4] Michel Abdalla and David Pointcheval: *Simple Password-Based Authenticated Key Protocols*, CT-RSA 2005, Lecture Notes in Computer Science 3376, pp. 191–208, Springer-Verlag, 2005.

[5] Mihir Bellare, David Pointcheval and Phillip Rogaway: *Authenticated Key Exchange Secure against Dictionary Attacks*, Eurocrypt 2000, Lecture Notes in Computer Science, Vol. 1807, pp. 139–155, Springer-Verlag, 2000.

[6] Federal Office for Information Security (BSI): *Advanced Security Mechanism for Machine Readable Travel Documents – Extended Access Control (EAC), Password Authenticated Connection Establishment (PACE), and Restricted Identification (RI)*, BSI-TR-03110, Version 2.0, 2008.

[7] Ran Canetti, Oded Goldreich and Shai Halevi: *The Random Oracle Methodology, Revisited*, STOC'98, pp. 209–218, ACM Press, 1998.

[8] Ran Canetti, Shai Halevi, Jonathan Katz, Yehuda Lindell and Philip D. MacKenzie: *Universally Composable Password-Based Key Exchange*, Eurocrypt 2005, Lecture Notes in Computer Science 3494, pp. 404–421, Springer-Verlag, 2005.

[9] Jean-Sebastien Coron, Jacques Patarin and Yannick Seurin: *The Random Oracle Model and the Ideal Cipher Model are Equivalent*, Crypto 2008, Lecture Notes in Computer Science 5157, pp. 1–20, Springer-Verlag, 2008.

[10] Marc Fischlin and Anja Lehmann: *Delayed-Key Message Authentication for Streams*, to appear at TCC 2010, Lecture Notes in Computer Science, Springer-Verlag, 2010.

[11] Oded Goldreich: *The Foundations of Cryptography*, Volume 1, Cambridge University Press, 2001.

[12] Oded Goldreich: *The Foundations of Cryptography*, Volume 2, Cambridge University Press, 2004.

[13] Thomas Icart: *How to Hash Into Elliptic Curves*, Crypto 2009, Lecture Notes in Computer Science, Springer-Verlag, 2009.

[14] Andrew Shallue and Christiaan van de Woestijne: *Construction of Rational Points on Elliptic Curves over Finite Fields.*, ANTS 2006, Lecture Notes in Computer Science 4076, pp. 510–524, Springer-Verlag, 2006.

[15] Victor Shoup: *Lower Bounds for Discrete Logarithms and Related Problems*, Eurocrypt'97, Lecture Notes in Computer Science 1233, pp. 256–266, Springer-Verlag, 1997.

[16] Michael Szydlo: *A Note on Chosen-Basis Decisional Diffie-Hellman Assumptions*, Financial Cryptography, Lecture Notes in Computer Science 4107, pp. 166–170, Springer-Verlag, 2006.

A Hardness of the PACE-DH Problem

Generic Hardness of the PACE-DH Problem. We show that the basic PACE-DH Problems are hard in Shoup's generic attack model. In Shoup's model the adversary's output can be represented by (linear) polynomials in the discrete logarithms of the input values $s_k, y_B, h = \log_G H$. (To simplify we consider s_i for $i \neq k$ to be fixed.) That is, Y_A is represented by a polynomial $F_{Y_A}(s_k, y_B, h) = \alpha + \beta s_k + \gamma y_B(s_k + h) + \delta h$ and so are the outputs K_i, by $F_{K_i}(s_k, y_B, h) = \alpha'_i + \beta'_i s_k + \gamma'_i y_B(s_k + h) + \delta'_i h$. In order to win the adversary's output must now be a DH value, i.e., the formal polynomial

$$\begin{aligned} F_{\mathrm{DH,i}}(s_k, y_B, h) &= y_B F_{Y_A}(s_k, y_B, h) - F_{K_i}(s_k, y_B, h) \\ &= -\alpha'_i - s_k\beta'_i - \delta'_i h + y_B\alpha + hy_B(\delta - \gamma'_i) + y_B s_k(\beta - \gamma'_i) + y_B^2 s_k\gamma + y_B^2 h\gamma \end{aligned}$$

must vanish for s_k, y_B, h. Note that $F_{\mathrm{DH,i}}(s_k, y_B, h)$ can only be the zero-polynomial if $\alpha = \gamma = \alpha'_i = \beta'_i = \delta'_i = 0$ and $\beta = \delta = \gamma'_i$ for known δ. But this means that the values Y_A and K_i are of the form $\delta(s_kG + H)$ and $\delta y_B(s_kG + H)$ and thus a trivial (linear) combination of the input data. Since the adversary is oblivious about k, i.e., Y_A is independent of k and all s_i's are distinct, the probability for the adversary guessing k correctly is at most $1/N$.

Now fix the value s_k as well. The probability that a polynomial $F_{\mathrm{DH,i}}$ of total degree now 2 vanishes for a random value is then at most $2m/q$, if m group operations have been performed (Shoup's model also takes into account a term of $O(m^2/q)$ for "accidental" collisions among group values). Hence, summing over all at most q_ℓ many K_i's the success probability of the adversary is at most $\frac{1}{N} + \frac{O(q_\ell m + m^2)}{q}$. Hence, to solve the problem generically with significant advantage one needs roughly $\sqrt{q}$ many operations or outputs.

Generic Hardness of the General PACE-DH Problem for DH2Point. Now consider the case that the element H is generated via a DH key agreement DH2Point where the adversary receives x_BG and then determines H by sending X_A. For a generic attacker $h = \log_G H$ itself is thus a polynomial of degree at most 2 in the variable x_B. (The case that the adversary goes first is a special case in which h is a polynomial of degree at most 1.) It therefore follows as above that a generic algorithm needs $\Omega(\sqrt{q})$ steps or outputs.

From the General PACE-DH Problem for DH2Point to PACE-DH. We next argue that, even though the adversary can potentially bias the outcome of the DH2Point execution, the only advantage of the adversary over the PACE-DH case of uniformly generators H must essentially stem from the ability to compute $\log_G H$. That is, assume that computing $\log_G H$ for the output of the honest party in an execution with an adversary $\mathcal{A}_{\mathsf{DH2Point}}$ against the gPACE-DH problem for DH2Point is infeasible (note that, otherwise, the gPACE-DH problem is obviously easy). Then we show that the gPACE-DH and the PACE-DH are equivalent (modulo the technical detail that the adversary against the gPACE-DH problem can forward some state information about the sampling process of H).

Denote by ϵ the non-negligible advantage of $\mathcal{A}_{\mathsf{DH2Point}}$ of breaking the gPACE-DH problem, i.e., the probability with which the adversary surpasses the pure guesssing probability. Call a fixed group element H_0 *good* (for the adversary) if the probability that the adversary's advantage exceeds $\epsilon/2$, given that $H = H_0$. Then we have that the probability that H, output by the honest user in an interaction of DH2Point with the adversary, is good, is at least $\epsilon/2$. Else the overall advantage of $\mathcal{A}_{\mathsf{DH2Point}}$ would be strictly smaller than ϵ (because the probablity of hitting a good H would then be less than $\epsilon/2$ and for bad H the adversary does not succeed with advantage more than $\epsilon/2$).

Define $\mathcal{H} \subseteq \langle G \rangle$ to be the set of all good H_0's. Then the set $\mathcal{H}$ must contain a non-negligible fraction of all group elements. Else the adversary can derive an H through DH2Point which lies in the set $\mathcal{H}$ with non-negligible probability $\epsilon/2$, and for which it can then compute the discrete logarithm by enumerating the discrete logarithms of all elements in $\mathcal{H}$ (and by checking for the right one). Since we assume that this is infeasible, it follows that $\mathcal{H}$ must form a non-negligible fraction of all elements.

We conclude that a uniformly distributed element H (as in the case of PACE-DH) lies in $\mathcal{H}$ with non-negligible probability. Note that for such H's the gPACE-DH adversary solves the problem with non-negligible probability. Hence, if, in addition, the gPACE-DH adversary does not forward any state information, then we derive that the adversary also solves the PACE-DH problem with non-negligible success probability.