Key-Secrecy of PACE with OTS/CafeOBJ *

Dominik Klein
Bundesamt für Sicherheit in der Informationstechnik (BSI)
Dominik.Klein@bsi.bund.de

Abstract

The ICAO-standardized Password Authenticated Connection Establishment (PACE) protocol is used all over the world to secure access to electronic passports. Key-secrecy of PACE is proven by first modeling it as an Observational Transition System (OTS) in CafeOBJ, and then proving invariant properties by induction.

1 Introduction

Cryptographic primitives, such as encryption mechanisms, hash functions or message authentication codes, undergo the scrutiny of a large community of researchers. While their mathematical foundations might not yet be understood in full detail, there have been few sudden groundbreaking attacks on them. Using these primitives as building blocks to construct security protocols is, however, another difficult challenge. In fact, despite using well-known cryptographic primitives, erroneous protocol specifications and design decisions have often lead to attacks. A famous example is [16], and the survey [7] contains an impressive list of failed attempts to design secure protocols. Formally proving properties of a protocol to exclude subtle attacks is one important step in the construction of security protocols.

Password Authenticated Connection Establishment (PACE) [4, 13] is a cryptographic protocol used all over the world for electronic passports. PACE establishes a secure communication channel between a terminal (trying to access data stored on the passport's RFID chip) and the passport itself. Ensuring trust in PACE is of uttermost importance due to several reasons: First, the predecessor of PACE, called Basic Access Control (BAC), is plagued with security concerns due to low-entropy passwords. Second, the contact-less RFID interface of electronic passports raises concerns of citizens that passports enable secret tracking or that criminals may remotely read out sensitive biometric information. Third, PACE is used in national id-cards that enable secure authentication for e-commerce.

*This is an author's version without proof reading. The final publication is available at Springer via http://dx.doi.org/10.1007/978-3-319-17581-2_11

CafeOBJ is an algebraic specification and programming language [8]. After specifying a formal model, e.g. of a cryptographic protocol such as PACE, CafeOBJ can also be used as an interactive theorem prover to show invariant properties of such a specified model: Mathematical proofs are written as *proof scores*, and a proof can be established by *executing* its proof score. This approach is used in this paper.

The contribution of this paper is threefold. First, key secrecy of PACE itself is shown, strengthening trust in the protocol. Second, while CafeOBJ has a proven track-record in the verification of security protocols [17, 18, 19, 20, 21, 23], the proof serves once more as a case study to show that theorem proving in CafeOBJ scales well beyond simple academic problems to real-world scenarios. Third, to the author's best knowledge, this proof is the first to model a protocol based on a Diffie-Hellman key-exchange in such detail in CafeOBJ. This might serve as a foundation for analyzing other DH-based protocols. The source code of the proof is available at `https://github.com/d-klein/ots-proof`.

The structure of this paper is as follows: In Section 2, the PACE protocol is introduced. A very brief recapitulation of modeling OTSs in CafeOBJ, and proving their invariants is given in Section 3. Section 4 provides an abstract version of PACE and shows how to model it as an OTS. The proof of key secrecy of PACE is shown in Section 5. Experiences and learned lessons are summarized in Section 6, and related work is reviewed in Section 7. Finally, concluding remarks are given in Section 8.

2 The PACE Key Agreement Protocol

To ensure compatibility with existing document formats and infrastructure, contactless RFID chips were chosen for electronic passports. This introduces two risks that need to be addressed: *Skimming*, i.e. an attacker reading out data from the passport without authorization, and *eavesdropping*, i.e. intercepting communication data during transmission. Note that skimming requires an online connection with the passport, whereas eavesdropped data can be analyzed offline after interception.

To prevent skimming, a terminal accessing data on the passport should prove that it is authorized to access the data. This can be done by e.g. reading information printed on the passport by OCR, and sending this data to the chip. The terminal thus demonstrates that it has physical access to the passport, and a passport holder can control electronic access to his passport by controlling physical access. Printed information on the passport often has low entropy. The machine-readable zone (MRZ) for example can be read by OCR and has 88 digits, but the vast majority of digits are not unique w.r.t. each passport, or can be easily guessed. Just hashing this printed data to directly derive a session key does not prevent sufficiently against offline attacks on eavesdropped transmission data, since the session key is the same for each session, and also has low entropy. Instead, a strong session key unique to each session is required to prevent (offline) analysis of eavesdropped transmission data.

The goal of the PACE key agreement protocol is to establish a secure, authenticated connection with a strong session key between the chip inside a passport and a corresponding terminal. PACE uses a pre-shared low entropy password to derive a strong session key by using a Diffie-Hellman key exchange [9]. The protocol is versatile in the sense that it allows to use either standard multiplicative groups of integers modulo p or groups based on elliptic curves. The latter is important in practice, since RFID chips have limited processing power.

The protocol works as follows: First, it is assumed that a common low entropy password π is known both by the chip and the terminal. Depending on the document type (international travel document, national id-card etc.) and use-case (border control, e-commerce) three solutions exist in practice: 1.) The password is derived from the MRZ, 2.) the password is derived from a Card Access Number (CAN) specifically printed on the document for this purpose or 3.) the password is derived from a secret personal identification number (PIN) known only to the owner of the document. In all cases, the password is stored on the chip in a protected way. To read out data on the chip, the MRZ is optically read by the terminal, or the CAN or the PIN is entered manually.

In the next step, the chip sends both a random nonce s encrypted by a symmetric cipher with the hash $\mathcal{H}$ of π and the domain parameter D_{PICC} for the group operation to the terminal. Using a *mapping function* and the domain parameter, the nonce s is mapped to some generator g of the group $\langle g \rangle$. Both the terminal and the chip chose another nonce x resp. y and compute exponents, i.e. the group operation is applied with the nonce together with the generator to derive g^x resp. g^y. These are then shared, and a key $K = (g^x)^y = (g^y)^x$ and MAC and session-keys are derived. Knowledge of the sent exponents and the key is verified by exchanging MAC-tokens. See Figure 1 for a brief overview of the protocol. For more detailed specifications, see [4].

3 OTS, CafeOBJ and Invariant-Proving

The PACE protocol is modeled as an Observational Transition System (OTS). For precise definitions and an introduction to OTSs, cf. [19]. Here, only a brief recapitulation on how OTSs are modeled in CafeOBJ is provided in order to give an intuition of the overall proof approach and proof structure. An OTS is a triple of a set of observable values, a set of initial states, and a set of conditional transition rules. A protocol can be modeled as an OTS, where in each state of the protocol, observations on this state can be made. The effect of a state change on the observations is described by transitions. An *invariant* is a property that holds (is observable) in all states reachable from the initial ones.

CafeOBJ is based on equational reasoning. Algebraic data types and operations on them are described by conditional rewrite rules. Rewrite rules are called equations in CafeOBJ, but they are applied directed from left to right. An OTS is modeled in CafeOBJ as follows:

- The state space is modeled as a *hidden sort* H.

Passport Chip (PICC)		Terminal (PCD)
	shared password π	
choose *nonce* $s \leftarrow \mathbb{Z}_q$ static domain parameter D_{PICC}		
$z = \mathbf{enc}(\mathcal{H}(\pi), s)$		
	$\xrightarrow{D_{\mathrm{PICC}}, z}$	
		$s = \mathbf{dec}(\mathcal{H}(\pi), z)$
$g = \mathbf{map}(D_{\mathrm{PICC}}, s)$		$g = \mathbf{map}(D_{\mathrm{PICC}}, s)$
choose $x \leftarrow \mathbb{Z}_q^*$		choose $y \leftarrow \mathbb{Z}_q^*$
$h_1 = g^x$		$h_2 = g^y$
	$\xrightarrow{h_1}$	
	$\xleftarrow{h_2}$	
abort, if $h_2 \notin \langle g \rangle$ or $h_1 \doteq h_2$		abort, if $h_2 \notin \langle g \rangle$ or $h_2 \doteq h_1$
$K = h_2^x = (g^y)^x$		$K = h_1^y = (g^x)^y$
$K_{\mathrm{MAC}} = \mathcal{H}(K\|\|1)$		$K_{\mathrm{MAC}} = \mathcal{H}(K\|\|1)$
$K_{\mathrm{ENC}} = \mathcal{H}(K\|\|2)$		$K_{\mathrm{ENC}} = \mathcal{H}(K\|\|2)$
$T_{\mathrm{PICC}} = \mathbf{mac}(K_{\mathrm{MAC}}, h_2)$		$T_{\mathrm{PCD}} = \mathbf{mac}(K_{\mathrm{MAC}}, h_1)$
	$\xrightarrow{T_{\mathrm{PICC}}}$	
	$\xleftarrow{T_{\mathrm{PCD}}}$	
abort, if $T_{\mathrm{PCD}} \neq \mathbf{mac}(K_{\mathrm{MAC}}, h_1)$		abort, if $T_{\mathrm{PICC}} \neq \mathbf{mac}(K_{\mathrm{MAC}}, h_2)$

Figure 1: The PACE protocol.

- A data type D is described in order-sorted algebra with *visible sort* V.
- An observation is modeled as a CafeOBJ behavioral operator:

  ```
  bop o : H V1 V2 ... VN -> V
  ```

  `V1,...,VN` and `V` are visible sorts corresponding to data types $D_1, \ldots D_n$, and `H` is the hidden sort representing the state space. Intuitively, this equation describes that the observation `V` can be made in state `H`, where `H` is characterized by `V1 ... VN`.
- A transition is also modeled as a CafeOBJ behavioral operator:

  ```
  bop t : H V1 V2 ... VM -> H
  ```

  The first argument of `t` refers to the current state. The operator `t` — identified by the indices `V1 ... VM` — maps the current state to another state in the state space. How this transition operator affects the state space in particular, is defined in CafeOBJ with conditional equations of the form:

```
ceq o(t(X,Y1,...,YM),Z1,...,ZN) = changeval(X,Y1,...,YM,Z1,...ZN)
        if effective-condition(X,Y1,...,YM,Z1,...,ZN) .

ceq t(X,Y1,...,YM) = X
        if not effective-condition(X,Y1,...,YM,Z1,...,ZN) .
```

Here `changeval` is the operation that changes values of the observation to the ones of the successor state, and `effective-condition` evaluates whether the condition to apply the transition is met in the current state. If the observed values never change when applying the transition, one can combine the above simply to:

```
eq o(t(X,Y1,...,YM),Z1,...,ZN) = o(X,Y1,...,YM).
```

CafeOBJ uses proof scores to prove invariants that hold in a model that is specified as described above. Proof scores define the proof obligations and induction hypothesis needed to proof invariants by induction.

Proof Scores.

A proof score of an invariant consists of two parts: First, the induction hypothesis w.r.t. the predicate in the initial state is shown. Then the induction step follows. For each invariant $\text{pred}_i(s, \mathbf{x})$ a corresponding operator and an equation is defined:

```
op invI : H V1 V2 ... VN -> Bool .
eq invI(S,X1,...,XN) = ... .
```

In the definitions of visible sorts in the specification, also a constant `init` is defined, denoting an arbitrary initial state. Then to prove $\text{pred}_i(s, \mathbf{x})$, one fixes arbitrary objects `v1,...,vN` for the visible sorts `V1,...,VN` and issues a reduce command w.r.t. the initial state: `red invI(init,v1,...,vN)`.

For the induction step one has to show that if $\text{pred}_i(s, \mathbf{x})$ holds in state s, then it also holds in any possible next state s'. For each predicate one fixes arbitrary states s and s' by `ops s,s' : -> H`, defines an operator of form `op istepI : V1 V2 ... VN -> Bool` and an equation for the induction step:

```
eq istepI(X1,X2,...,XN) = invI(s,X1,,...,XN) implies invI(s',X1,...,XN) .
```

Then one fixes arbitrary objects `v1,...,vN` for the visible sorts, defines how `s'` results from `s` by a transition `t` by `eq s' = t(s,...) .`, and issue a reduce command `red istepI(v1,...,vN)`. The reduce command uses the equations to obtain the equational normal form of an expression. If both for the initial state and the induction step rewriting to normal form reaches the constant `true`, the proof w.r.t. to transition `t` has succeeded. For a full proof, all defined transitions have to be considered.

Lemmata.

Quite often the induction step cannot be shown directly, since the induction hypothesis is too weak. Then a lemma is needed. Let `invJ` be a predicate with free variables of visible sorts `E1,...,EK`, and let `e1,...,eK` denote either free variables of, or expressions (i.e. terms) of these sorts. One can strengthen the induction hypothesis by augmenting `invJ` in state `s`, i.e. by issuing `red invJ(s,e1,...eK) implies istepI(v1,...,vN)`. One advantage in OTS/CafeOBJ is that one can use `invJ` to strengthen the induction step in the proof of `invI` and vice-versa.

Case Analysis.

Another proof technique is case analysis. Suppose for example that `v1` is assumed to be of arbitrary form. For a constructor `f`, we can then distinguish the case that either `v1` is constructed by `f` applied to some arbitrary `vC`, or that this is not the case. Then the induction step is split: One declares `v1 = f(vC)`, and reduces `red istepI(v1,...)`. Then one does the same again, but declares `(v1 = f(vC)) = false` before reducing. Clearly all possible cases have been exhaustively considered, since it is always true that:

```
(v1 = f(vC)) or (not (v1 = f(vC)))
```

Of course it is possible to strengthen the induction hypothesis by more than one predicate, and to combine lemma application with case analysis.

4 Modeling PACE in CafeOBJ

The system is modeled in a way such that an unbounded number of principals interact with each other by sending messages. Honest principals behave according to protocol. Malicious ones can fake and forge messages. The malicious principals are modeled as the most general intruder according to the Dolev-Yao intruder model [10]. Moreover the following assumption are made:

1. Cryptographic primitives are sound. Random nonces are unique and cannot be guessed, encrypted messages can only be decoded by knowing the correct key, hashes are one-way and there are no collisions, and two message authentication codes are the same only if generated from the same message with the same key.

2. The intruder can glean any public information (i.e. messages, ciphers etc.) that is sent in the network.

3. The intruder can send two kinds of messages: He can use ciphers based on cryptographic primitives from existing messages as black boxes to send new fake messages, or he can use eavesdropped information to generate new messages from scratch. But, as noted above, he cannot eavesdrop information from ciphers based on cryptographic primitives without knowing the corresponding keys or passwords.

4.1 An abstract version of PACE

To abstract away from implementation-dependent information and those that cannot be captured in the Dolev-Yao model anyway, the following abstract version of the PACE protocol is used.

$$
\begin{array}{lll}
\text{Message 1}: & p \to q: & \mathbf{enc}_{\pi}(n_s, D) \\
\text{Message 2}: & p \to q: & *(n_a, G) \\
\text{Message 3}: & q \to p: & *(n_b, G) \\
\text{Message 4}: & p \to q: & \mathbf{mac}(\mathcal{H}(*(n_a, *(n_b, G))), *(n_b, G), D) \\
\text{Message 5}: & q \to p: & \mathbf{mac}(\mathcal{H}(*(n_b, *(n_a, G))), *(n_a, G), D)
\end{array}
$$

It is assumed that a run of PACE is conducted by exchanging five messages. In the first step, a message is sent from a principal p to another one q. The message encrypts a random nonce n_s with the shared password π, with attached static domain parameters D. Next, p maps the nonce n_s from the first message with the domain parameters to a group generator G. Then p chooses a random nonce n_a, applies the operator $*$ to both n_a and G and sends the result $*(n_a, G)$ to q. In a similar manner, q chooses a random nonce n_b and sends $*(n_b, G)$ to p. Next, p computes the key $\mathcal{H}(*(n_a, *(n_b, G)))$. He then sends a message authentication code — encoded with that key — with the received exponent $*(n_b, G)$ and domain parameters D to q, in order to verify knowledge of both the received exponent and the generated key. Principal q does the same in reverse, and the common key $\mathcal{H}(*(n_a, *(n_b, G)))$ is used from now on to exchange encrypted messages.

4.2 Basic Data Types

The following algebraic data types, i.e. visible sorts and corresponding constructors are used:

- `Principal` denotes both honest and malicious principals in the network.
- `Random` denotes random nonces. Random nonces are supposed to be unique and unguessable.
- `Dompar` denotes the static domain parameters of PACE. Used domain parameters are not secret and known to every principal.
- `Mappoint` denotes a group generator. The constructor `maptopoint` of data type `Mappoint` takes as input a random nonce and static domain parameters and returns a group generator. It is supposed that `maptopoint` is a one-way function.
- `Expo` denotes an exponent of the form g^x, where the group generator g is generated by `maptopoint` using a random nonce and domain parameters as input.

- `Hash` denotes keys — it is supposed that hashing is the key derivation function. The constructor `hash` takes as input a random nonce and an exponent and returns a key.
- `Cipher1` denotes the cipher resulting from a symmetric encryption. Its constructor `enc` takes as input a random nonce and static domain parameters. It is implicitly assumed that a `Cipher1` is encoded with the shared password π in the following way: Given a `Cipher1`, every principal is able reconstruct the static domain parameters. But only if he knows the shared password π, he is able to decode the random nonce.
- `Cipher3` denotes message authentication codes. The constructor `mac` takes as input a hash, an exponent and domain parameters.

Three sorts and data types are defined for the messages in Section 4: Message 1 of Section 4 is of type `Message1`, messages 2 and 3 are of type `Message2`, and messages 4 and 5 are of type `Message3`. Here, `Message1` is a `Cipher1` attached with meta-information describing the creator, the (seemingly) sender, and the receiver of a message. For example

```
me1(intruder,p,q,c)
```

denotes a `Message1` where `c` is a `Cipher1`, and the message is (seemingly) sent from principal `p` to `q`, but was actually created by the intruder, i.e. faked and injected in the network. Similar, a `Message3` is a `Cipher3` attached with corresponding meta-information. The data type `Message2` is constructed by attaching meta-information to an exponent. Moreover for the definition of the data structures two design decisions — cf. also Section 6 — should be noticed:

4.2.1 Modeling of the shared password π.

PACE assumes a fixed shared password π known among honest principals. Knowledge of the password is modeled by a predicate `knowspi` where one sets `knowspi(intruder) = false`. No specific data type is introduced for decryption of messages of type 1, instead it is just distinguished between messages that are created by an honest principal who does know π and the intruder, who does not.

4.2.2 Equality of hashes.

The equality operator `_=_` for `hashes` is defined as

```
eq (H1 = H2) = (rand(H1) = rand(H2) and expo(H1) = expo(H2))
       or      (     rand(H1) = rand(expo(H2))
                 and rand(H2) = rand(expo(H1))
                 and point(expo(H1)) = point(expo(H2))) .
```

i.e. that $\mathcal{H}(*(n_a, *(n_b, G_1))) = \mathcal{H}(*(n_c, *(n_d, G_2)))$ if $G_1 = G_2$, $n_a = n_c$, $n_b = n_d$, or $G_1 = G_2$, $n_a = n_d$, $n_b = n_c$. This captures the equality of the keys generated during the key exchange.

4.3 Protocol Modeling

In order to collect all sent messages, all generated random nonces, and other information, the following definition of a *multiset* on an abstract level from [19] is reused. This definition is then later used as a parametrized module to define multisets containing the data-types defined in the previous section.

```
mod* SOUP (D :: EQTRIV) principal-sort Soup {
  [Elt.D < Soup]
  op empty : -> Soup {constr}
  op _ _ : Soup Soup -> Soup {constr assoc comm id: empty}
  op _\in_ : Elt.D Soup -> Bool
  var S : Soup
  vars E1 E2 : Elt.D
  eq E1 \in empty = false .
  eq E1 \in (E2 S) = (E1 = E2) or E1 \in S .
}
```

The operator `\in` defines membership in the multiset, and a space defines insertion. To collect all random nonces for example, one can define an observation `bop rands : System -> RandSoup` that takes as input a state, and returns as the observation a soup of random nonces. Given a random nonce `r` and a state `s`, one can test membership by `r \in rands(s)`, and — for example describing the effects of a transition — insert `r` in the multiset by `r rands(s)`. Observations and transitions are defined as follows:

```
-- observations
bop network  : System -> Network
bop rands    : System -> RandSoup
bop hashes   : System -> HashSoup
bop randsi   : System -> RandSoup
bop expos    : System -> ExpoSoup
bop cipher1s : System -> Cipher1Soup
bop cipher3s : System -> Cipher3Soup
-- transitions
bop sdm1  : System Principal Principal Random Dompar           -> System
bop sdm2  : System Principal Principal Random Message1         -> System
bop sdm3  : System Principal Principal Message1 Message2 Message2
                                                               -> System
-- faking and forging messages based on the gleaned info
bop fkm11 : System Principal Principal Cipher1                 -> System
bop fkm12 : System Principal Principal Random Dompar           -> System
bop fkm21 : System Principal Principal Expo                    -> System
bop fkm22 : System Principal Principal Random Random Dompar    -> System
bop fkm31 : System Principal Principal Cipher3                 -> System
bop fkm32 : System Principal Principal Random Expo Expo Dompar -> System
```

Seven observers are used to collect information:

- `network` returns a multiset of *all* messages that have been sent so far.

- `rands` returns a multiset containing *all* random nonces that have been generated so far.
- `hashes` returns all *keys* resulting from the PACE protocol that have been gleaned or self-generated by the *intruder*. The name stems from the fact that one considers `hash` to be the key derivation function.
- `randsi` contains all *random nonces* gleaned or self-generated by the *intruder*.
- `expos` contains all exponents that have been inserted in the network and
- `cipher1s` and `cipher3s` collect *all* ciphertexts of messages of type 1 and messages of type 3 (i.e. mac-tokens).

The transitions `sdm1`, `sdm2`, and `sdm3` describe state transitions and their effects on observations when an honest principal sends a message of type 1, 2 or 3. Therefore the conditions on when these transitions are effective, capture precisely the behavior of an honest principal. For example `sdm1` is defined as:

```
eq c-sdm1(S,P,Q,R,D) = not(R \in rands(S)) .
ceq network(sdm1(S,P,Q,R,D))  = me1(P,P,Q,enc(R,D)) network(S)
    if c-sdm1(S,P,Q,R,D) .
```

Thus an honest principal p can add a message `me1(P,P,Q,enc(R,D))` in state `S` — in message protocol notation $p \to q : \mathbf{enc}_\pi(R, D)$ — only to the network if the nonce `R` is fresh. Freshness means that `R` is not contained in the set of all nonces that have been generated before reaching state `S`. This freshness condition is modeled by the first equation.

The transitions `fkmXY` describe state transitions and their effects on observations when the intruder generates messages. Here one distinguishes two cases: 1.) the intruder fakes an existing message by changing its source and destination (`fkmX1`) and 2.) the intruder injects a new message in the network using information available to him (`fkmX2`). Therefore the effective conditions for these transitions are usually more lax than the ones for `sdmX`. For example the condition to fake a message of type 1

```
eq c-fkm11(S,P,Q,C1) = C1 \in cipher1s(S) .
```

is just that a cipher1 exists in the network. The intruder can then inject the message `me1(intruder,P,Q,C1)` with arbitrary source `P` and destination `Q`. Note that the meta information denoting the creator of the message cannot be altered by the intruder.

An example for the second case is the condition to construct an arbitrary new message of type 1

```
eq c-fkm12(S,P,Q,R,D) = (not (R \in rands(S))) or (R \in randsi(S)) .
```

Here the intruder can choose to either use a fresh random nonce, or one that he has gleaned or generated in an earlier state. He then injects the message `me1(intruder,P,Q,enc(R,D))` into the network.

5 Proving Key-Secrecy

Key secrecy is shown in the following sense: Suppose that one takes the perspective of an honest principal, i.e. one is either the passport or the terminal, and one behaves according to protocol. In particular it is assumed that

1. one has either sent a `Message1` with a nonce encrypted with the shared password π and domain parameters (passport) *or* one has received a `Message1` from a principal who knows π and decrypted it (terminal) and
2. one constructed a generator of the group with the nonce and the domain parameters from the above message, used the generator together with a fresh nonce to create an exponent, and sent it to the other party and
3. one *seemingly* (it is unknown who created the message) received an exponent back from that other party and
4. one *seemingly* received a MAC-token that, using ones secret nonce together with the received exponent as a key, validates that the other party knows ones sent exponent and the domain parameters.

Then the resulting key must *never* be known to the intruder. This can be almost verbatim translated into the next main theorem:

```
eq inv900(S,M1,M21,M22,M3,P,Q) =
  (M1 \in network(S) and M21 \in network(S)
   and M22 \in network(S) and M3 \in network(S)
   and sender(M3) = Q and receiver(M3) = P
   and creator(M21) = P and sender(M21) = P and receiver(M21) = Q
   and sender(M22) = Q and receiver(M22) = P
   and (not (creator(M21) = creator(M3)))
   and (not (P = Q)) and knowspi(P)
   and ((sender(M1) = P and creator(M1) = P and receiver(M1) = Q) or
        (sender(M1) = Q and receiver(M1) = P and knowspi(creator(M1))))
   and expo(M21) = expo(cipher3(M3))
   and dpar(cipher1(M1)) = dpar(point(expo(M21)))
   and rand(cipher1(M1)) = rand(point(expo(M21)))
   and dpar(cipher1(M1)) = dpar(cipher3(M3))
   and hash(cipher3(M3)) = hash(rand(expo(M21)),expo(M22)))
 implies
   not (hash(cipher3(M3)) \in hashes(S)) .
```

5.0.1 Application of Lemmata and Case Analysis.

To prove key secrecy one needs additional invariants. Central to strengthening the induction hypothesis for `istep900` is the invariant that the assumptions of `inv900` imply that both principals have implicitly agreed upon the same generator g, which itself depends on the nonce exchanged in the first message. For brevity suppose that `assump(S,M1,M21,M22,P,Q)` is a predicate that denotes truth of the assumptions of invariant `inv900` above. The invariant can then be expressed as:

```
eq inv800(S,M1,M21,M22,M3,P,Q) = assump(S,M1,M21,M22,P,Q)
  implies rand(point(expo(M22))) = rand(point(expo(M21))) .
```

How such a lemma is used in the proof together with case analysis is illustrated, albeit for a simpler invariant. Frequent use of the following invariant as a lemma for others is made. It states that if one is in a state `S`, and a `M1` of type `Message1` is in the network, then the random nonce of `M1` has been used and is thus included in the collection of all random nonces `rands(S)`.

```
eq inv300(S,M1) = M1 \in network(S)
                  implies rand(cipher1(M1)) \in rands(S) .
```

`inv300` is proven inductively on the number of transitions. In the case of transition `fkm11` one performs case analysis w.r.t. its effective condition:

```
(c-fkm11(s,p10,q10,c11) = false) or (c-fkm11(s,p10,q10,c11) = true)
```

Here `p10` and `q10` denote arbitrary principals, and `c11` denotes an arbitrary `cipher1`. For the first case, the proof directly succeeds:

```
open ISTEP
 ops p10 q10 : -> Principal .
 op m10 : -> Message1 .
 op c11 : -> Cipher1 .
 eq c-fkm11(s,p10,q10,c10) = false .
 eq s' = fkm11(s,p10,q10,r10,d10) .
 red istep300(m10) .
close
```

For the second case `c-fkm11(s,p10,q10,c11) = true`, one replaces the term with its definition `c11 \in cipher1s(s) = true` and performs another case analysis w.r.t. the equality `m10 = me1(intruder,p10,q10,c11)`.

```
open ISTEP
 ops p10 q10 : -> Principal .
 ops m10 : -> Message1 .
 op c11 : -> Cipher1 .
 eq c11 \in cipher1s(s) = true .
 eq m10 = me1(intruder,p10,q10,c11) .
 eq s' = fkm11(s,p10,q10,c11) .
 ***
close
```

If one directly tries to prove the induction step by reducing `red istep300(m10)` inserted at `***`, CafeOBJ outputs

```
rand(c11) \in rands(s) xor
  me1(intruder,p10,q10,c11) \in network(s) xor ...
```

This indicates that if `me1(intruder,p10,q10,c11)` is not already included in and thus inserted in the network as a result of the transition `fkm11`, then

`rand(c11) \in rands(s)` must be true for the induction step to hold. Therefore the induction hypothesis needs to be strengthened. One does so by introducing yet another invariant `inv150`, which states that if a `cipher1` is in the network, than its random nonce is included in the set of all used random nonces.

```
eq inv150(S,C1) = C1 \in cipher1s(S) implies rand(C1) \in rands(S) .
```

And indeed, applying `inv150` as a lemma at `***` by inserting

```
red inv150(s,c11) implies istep300(m10)
```

successfully finishes the induction step. Therefore it has been verified that if a `Cipher1` exists, i.e. is included in the set of collected ciphers observable in state `S` in the network, then the random nonce of that cipher must be included in the set of collected nonces observable in that state.

6 Experience and Lessons Learned

From the experience of applying OTS/CafeOBJ to a rather large real-world example, three guidelines are formulated:

1. *Refine your specification.* When stuck in a proof attempt, it is worthwhile to reconsider the specification. Take for example the definition of equality of hashes. Initially equality was defined for two ciphers3's `C1` and `C2` intuitively as

   ```
   eq (C1 = C2) = (hash(C1) = hash(C2) and expo(C1) = expo(C2)
                   and dpar(C1) = dpar(C2)) .
   ```

   This has the awkward consequence that messages can no longer uniquely be identified: When a principal sends a message of type 3, implicitly *two* messages are added to the network, one w.r.t. each case of equality of the hash of the cipher. Then for example an invariant like

   ```
   m3 \in network(s) implies cipher3(m3) \in cipher3s(s)
   ```

   does not hold if we have

   ```
   cipher3(m3) = mac(hash(r2,expo(r1,...),...)
   ```

   and

   ```
   mac(hash(r1,expo(r2,...),...) \in cipher3s(s).
   ```

   This makes reasoning during the induction steps quite unintuitive and led to defining equality of cipher3's as syntactic equality of normals forms, and formulating theorems accordingly when referring to multiple cipher3's with the same hash.

2. *Simplify your specification.* Trying to specify every detail naturally gives a proof that is most faithful to the real protocol. It however also leads to more involved proofs and case-analysis. For example, we purposely decided not to fully model the symmetric cipher used to encrypt the shared password π, but rather to model knowledge of π with a predicate.

3. *A deductive proof approach.* It is very simple in CafeOBJ to quickly add a lemma without proving it. Some invariants, like `inv900` in the current case, are quite involved, and it is likely that one encounters problems with the specification during the proof, and refines or simplifies the specification thereafter. This often also affects helper lemmata. It it thus very useful to focus on the proof of a complex invariant, thereby using several simpler, unproven lemmata, and only afterwards focus on the proof of the latter.

The main hindrance when conducting the proof is related to performance. Suppose one is proving an invariant of the form $a_1 \wedge a_2 \ldots \wedge a_n \implies b$, such as `inv900`. A direct proof attempt often does not terminate, due to the amount of branching. To get a terminating result, one can make a trivial case analysis w.r.t. a_i, e.g. distinguish the case for $\neg a_1$, for $a_1 \wedge \neg a_2$, and so on, to finally reach the case for $a_1 \wedge \ldots \wedge a_n$. Even then sometimes a proof attempt does not terminate, so additional assumptions and corresponding cases have to be added. Almost all cases are trivial – it is obvious that in the case with the assumption $\neg a_1$ the above invariant holds – but lead to a blow up of the size of the proofs. For example, our proof score consists of 38427 lines, of which the vast majority are for such trivial cases. Fortunately, the majority of these cases could be generated automatically by scripts. Nevertheless, tools that tie more directly with CafeOBJ, or come distributed with it, would be certainly helpful for an easier work-flow and increased productivity.

All in all, 40 invariants of the formalization of PACE were proven. The verification of all invariants together takes approximately two hours and eight minutes on an Intel Core i7-3520M @ 2.9 Ghz.

7 Related Work

7.0.2 Security Analysis of PACE.

An inductive verification [5] of the PACE protocol has been conducted in the verification support environment (VSE) [14]. VSE has been developed in the 1990's by a consortium of German universities and industry to provide a tool to meet industry needs for the development of highly trustworthy systems. Since the proof source is not publicly published and the VSE tool and documentation is not available for download, a comparison is difficult. An independent verification of the proof however is important to ensure trust in the protocol, not only for users, but also for work in international standardization bodies. A pen-and-paper proof for security in the sense of Abdalla, Fouque and Pointcheval [1] has been given in [2]. In [6] attempts are made to merge the pen-and-paper proof with the VSE-proof.

7.0.3 Formal Analysis of Security Protocols.

According to [4], the execution of the protocol, and thus the state space, is not bounded. An approach based on model-checking seems therefore not appropriate. Other than (classical) model-checking, a plethora of tools and approaches exist to formally analyze security protocols, and the reader is referred to [3] for a comprehensive overview. Compared to other tools, the choice of CafeOBJ was motivated rather from the perspective of a practitioner, and not necessarily due to other tools lacking features. In particular the OTS/CafeOBJ approach is well documented, has a proven track record w.r.t. security protocol verification [17, 18, 19, 20, 21, 23], the CafeOBJ platform is very stable, and modeling of protocols is straight-forward. Also, it is not difficult to start with an abstract specification, and then add details and extend proofs later on.

The lack of automation in OTS/CafeOBJ is a double-edged sword. On one hand no hidden limitations exist, whereas most tools that aim for full automation make some assumptions to e.g. reduce the state space. It is sometimes not easy to anticipate in advance which of these limitations apply for the protocol one intends to prove. Moreover the manual approach forces oneself to recapitulate on the formalization and its appropriateness of capturing the protocol in question. On the other hand, the lack of automation is sometimes not time-effective and somewhat tedious. Constructing tools that not only offer a high level of automation, but also fully axiomatize Abelian group-theory to account for more in-depth algebraic attacks is an ongoing research-topic, with several tools, e.g. the Tamarin tool [24], which is based on multiset rewriting, or an extended version of ProVerif [15]. Maude, another member of the OBJ family, has been used for formal analysis of security protocols [22], and in particular the Maude-NPA [11] tool offers a narrowing based approach for Diffie-Hellman. Last, automation of the OTS/CafeOBJ approach itself has also recently been increased significantly [12].

All these approaches are natural candidates when extending the proof, e.g. by adding detail to the specification w.r.t. mapping a point and domain parameters to a group generator, or when extending to the protocol sequence to the full protocol sequence for extended access control

8 Conclusion and Future Work

Key secrecy has been successfully verified in CafeOBJ. This not only facilitates trust in the PACE protocol, but also represents one more case-study that shows that the OTS/CafeOBJ approach scales well beyond toy-examples like NS(L)PK to real-world scenarios. Also, the PACE proof can serve as a guide on how to model a DH-key exchange in CafeOBJ. Key-Secrecy however, is only one important property of PACE. Future directions include to extend the proof to mutual authentication, perfect forward secrecy, and the full EAC2 protocol stack, possibly with the help of the automated tools mentioned in Section 7.

References

[1] Abdalla, M., Fouque, P.A., Pointcheval, D.: Password-based authenticated key exchange in the three-party setting. In: Proc. 8th PKC. LNCS, vol. 3386 (2005)

[2] Bender, J., Fischlin, M., Kügler, D.: Security analysis of the PACE key-agreement protocol. In: ISC. LNCS, vol. 5735 (2009)

[3] Blanchet, B.: Security protocol verification: Symbolic and computational models. In: Proc. 1st POST. LNCS, vol. 7215 (2012)

[4] BSI: Advanced security mechanisms for machine readable travel documents (2012)

[5] Cheikhrouhou, L., Stephan, W.: Meilensteinreport: Inductive verification of PACE. Tech. rep., DFKI GmbH (2010)

[6] Cheikhrouhou, L., Stephan, W., Dagdelen, Ö., Fischlin, M., Ullmann, M.: Merging the cryptographic security analysis and the algebraic-logic security proof of PACE. In: Sicherheit. LNI, vol. 195 (2012)

[7] Clark, J., Jacob, J.: A survey of authentication protocol literature (1997)

[8] Diaconescu, R., Futatsugi, K.: CafeOBJ Report: The language, proof techniques and methodologies for object-oriented algebraic specification, AMAST Series in Computing, vol. 6. World Scientific (1998)

[9] Diffie, W., Hellman, M.E.: New directions in cryptography. IEEE Transactions on Information Theory 22(6) (1976)

[10] Dolev, D., Yao, A.C.C.: On the security of public key protocols. IEEE Transactions on Information Theory 29(2) (1983)

[11] Escobar, S., Meadows, C., Meseguer, J.: Maude-NPA: Cryptographic protocol analysis modulo equational properties. In: FOSAD. lncs, vol. 5705 (2007)

[12] Găină, D., Zhang, M., Chiba, Y., Arimoto, Y.: Constructor-based inductive theorem prover. In: 5th CALCO. LNCS, vol. 8089 (2013)

[13] ICAO: Doc 9303 – Machine readable travel documents

[14] Koch, F.A., Ullmann, M., Wittmann, S.: Verification support environment. In: Proc. 8th CAV. LNCS, vol. 1102 (1996)

[15] Küsters, R., Truderung, T.: Using ProVerif to analyze protocols with Diffie-Hellman exponentiation. In: Proc. 22nd CSF (2009)

[16] Lowe, G.: An attack on the Needham-Schroeder public-key authentication protocol. Inf. Process. Lett. 56(3) (1995)

[17] Ogata, K., Futatsugi, K.: Rewriting-based verification of authentication protocols. Electr. Notes Theor. Comput. Sci. 71 (2002)

[18] Ogata, K., Futatsugi, K.: Flaw and modification of the iKP electronic payment protocols. Inf. Process. Lett. 86(2) (2003)

[19] Ogata, K., Futatsugi, K.: Proof scores in the OTS/CafeOBJ method. In: Proc. 6th FMOODS 2003. LNCS, vol. 2884 (2003)

[20] Ogata, K., Futatsugi, K.: Equational approach to formal analysis of TLS. In: Proc. 25th ICDCS. IEEE Computer Society (2005)

[21] Ogata, K., Futatsugi, K.: Proof score approach to analysis of electronic commerce protocols. International Journal of Software Engineering and Knowledge Engineering 20(2) (2010)

[22] Ölveczky, P.C., Grimeland, M.: Formal analysis of time-dependent cryptographic protocols in real-time maude. In: Proc. 21st IPDPS (2007)

[23] Ouranos, I., Ogata, K., Stefaneas, P.S.: Formal analysis of TESLA protocol in the timed OTS/CafeOBJ method. In: Proc. 5th ISoLA. LNCS, vol. 7610 (2012)

[24] Schmidt, B., Meier, S., Cremers, C.J.F., Basin, D.A.: Automated analysis of Diffie-Hellman protocols and advanced security properties. In: Proc. 25th CSF (2012)