Formal Analysis of the Estonian Mobile-ID Protocol

Peeter Laud[1,2] and Meelis Roos[1,2]

[1] Cybernetica AS
[2] Tartu University, Institute of Computer Science

Abstract. In this paper, we report the results of the formal analysis performed on the Estonian Mobile-ID protocol (deployed since 2008), allowing citizens and permanent residents of Estonia to authenticate themselves and issue digital signatures with the help of a signature-capable SIM-card inside their mobile phone. We analyze the resiliency of the protocol to network attacks under various threat models (compromised infrastructure, client application, etc., confusing user interface) and give suggestions for improvement.

1 Introduction

Since 2002, Estonia has issued chipped ID cards to its citizens and permanent residents. The card has been integrated into a national public-key infrastructure. Upon the initialization of a new ID card for the user U, two RSA keypairs are loaded into it. The card is capable of performing modular exponentiations with the secret exponents of those keypairs. During initialization, certificates binding the public keys to the user U are also issued and stored on the card (as well as in a public database). The certificates are issued by a certification authority (CA) in the list of state-recognized CAs. The intended uses for the secret keys (as recorded in certificates) are identification (for the first keypair) and signing (for the second keypair).

The identification functionality of the card can be used when accessing public web-sites. When the user has directed his client application (usually a web browser) to access a server over a secured connection, the two will perform a TLS handshake [10] during which both the server and the client are authenticated. During the protocol, the client has to sign a message, a hash of which is handed over to the ID card in a smartcard reader connected to client's computer. The card will apply the RSA exponentiation to this hash, using the secret exponent in the first RSA keypair. The result is handed back to the client application which includes it in a protocol message. To activate the card's signing functionality, a PIN (consisting of four or more decimal digits) has to be given to it (different PINs for different keypairs). The PIN is entered either from the computer keyboard or the PIN-pad integrated with the card reader. In the first case, the PIN is handled by the client application and given to the card together with the hash to be signed. The card locks up after a couple consequtive incorrect enterings of the PIN.

Since 2007, Eesti Mobiiltelefon (the largest Estonian mobile operator) in cooperation with Sertifitseerimiskeskus AS (the only state-recognized CA in Estonia) have issued mobile SIM cards with the same functionality [15]. Later that year, they were joined by the Lithuanian mobile operator Omnitel [20]. Similarly, RSA keypairs are loaded into those cards and the public keys are issued certificates binding them with users. The SIM card can compute signatures on users' behalf after being given a PIN which is entered from the keypad of the mobile phone. The mobile ID thus reduces the threats related to handing over one's PIN to a possibly trojaned computer. Trojan horse attacks on mobile phones are as of now only a negligible part of the malware market [13], although should their number and impact increase, the conclusions made in this paper must be reconsidered. Another claimed advantage of mobile ID is convenience — no smartcard reader is necessary [16].

At the same time, client authentication in the Mobile-ID protocol uses a much more complex protocol than the TLS handshake, and involves many more parties. This raises a number of interesting security questions. The goal of the research reported in this paper was to formalize the Mobile-ID protocol in the protocol checker ProVerif [7] and use it to explore what happens if various parts of the system are acting differently than expected. We also have tested the implementation of central parts of the protocol; this paper shows how to formally model the possible weaknesses we found. After reporting the results of this exploration, we also suggest modifications for the protocol to make it more secure under certain attacks.

A general security analysis of Mobile-ID has been performed previously [21]. This analysis was considerably broader in scope than the one reported in this paper; it considered not just the network attacks, but also legal and human issues and risks related to the failure of technical components. The conclusion of the analysis was that generally, the risks associated with Mobile ID are the same as the risks of using the ID card. There are some additional risks related to the necessity to trust the extra infrastructure used in the Mobile ID protocols. No formal analysis of the protocols was presented in [21].

In this paper we first describe the Mobile-ID authentication protocol and base security assumptions (honesty of certain parties and security of certain channels) for it. The base security assumptions describe the situation where only parties that are normally considered to be dishonest can be dishonest. As next we describe how we have formalized the Mobile-ID protocol in ProVerif. In the next section we describe the results of our analysis. We have analyzed the protocol under base security assumptions, as well as several different, stronger assumptions where we have allowed certain parties or connections to be under adversarial control. We finish the paper with suggestions for improving the protocol, as well as general conclusions.

2 The Mobile-ID Protocol

The Mobile-ID protocol [4] involves the following parties:

- The user U that wants to access some service requiring authentication.
- The phone P of that user, as well as the SIM card inside it. Although technically two different units, we model them as a single one. The SIM card knows the secret signing key sk_U, the corresponding public key pk_U of which is bound to the identity of U by the certificate cert_U.
- The client application C of that user, typically a computer running a web browser. The client application is used to actually access the service.
- The server S that the user wants to connect to. It has a secret key sk_S which public counterpart pk_S is bound to the name S by the certificate cert_S. With the help of sk_S, the server can participate in a TLS handshake as a server.
- The mobile operator O that has issued the SIM-card of the phone P.
- The DigiDocService D [4]. This is a central party of the protocol meditating the authentication process and forwarding the messages to right parties. The DigiDocService has a secret key sk_D allowing it to participate in a TLS handshake as a server. The certificate cert_D binds the corresponding public key pk_D to the identity of D.

The parties above actively take part of a protocol session. Besides them, there is also the certificate authority CA issuing the certificates. Also, there are means (OCSP) to verify the status of a certificate [19].

The Mobile-ID protocol [4] is depicted in Fig. 1. A protocol session is initiated by the user U deciding to contact the server S and informing the client application C about this choice. The client application locates the certificate cert_S of the server and initiates a TLS handshake with it. During the handshake, the server is authenticated to the client, but not vice versa. The resulting TLS tunnel is used to communicate the rest of the messages between C and S. To authenticate the user, C sends to S the name U (which also determines the phone number P). The server S then initiates a TLS handshake with D, again resulting in the secure identification of D, but not S. Again, the TLS tunnel is used to encapsulate the messages between S and D. The server S sends to D the names U and P, identifying the user. Additionally, S generates a 10-byte challenge r_1 (part of the challenge signed with sk_U) and sends it to D, too. Optionally, r_1 may be empty. Also, S sends to D something that identifies itself: $\tilde{S} = (S, m)$ where m is an additional message that will be displayed to the user on the screen of the mobile phone. Both S and D will then locate the certificate cert_U of the user.

The DigiDocService will generate a random nonce r_2. The phone of the user is supposed to sign the concatenation of r_1 and r_2 with the key sk_U, where pk_U is included in cert_U. DigiDocService forwards both $\tilde{S}$ and $r_1 \| r_2$ to the phone P via the mobile operator O. The communication between D and O is protected by a VPN solution. The communication between O and P is through SMS-s, and is protected by encrypting the messages between O and P with the symmetric key $\tilde{K}_P$ known only to themselves. DigiDocService also computes CC_1 as the *control code* of $r_1 \| r_2$. The control code consists of four decimal digits. The control code CC_1 is forwarded to the client application C that displays it to the user U. The phone P also computes the control code of $r_1 \| r_2$ and displays it to the user,

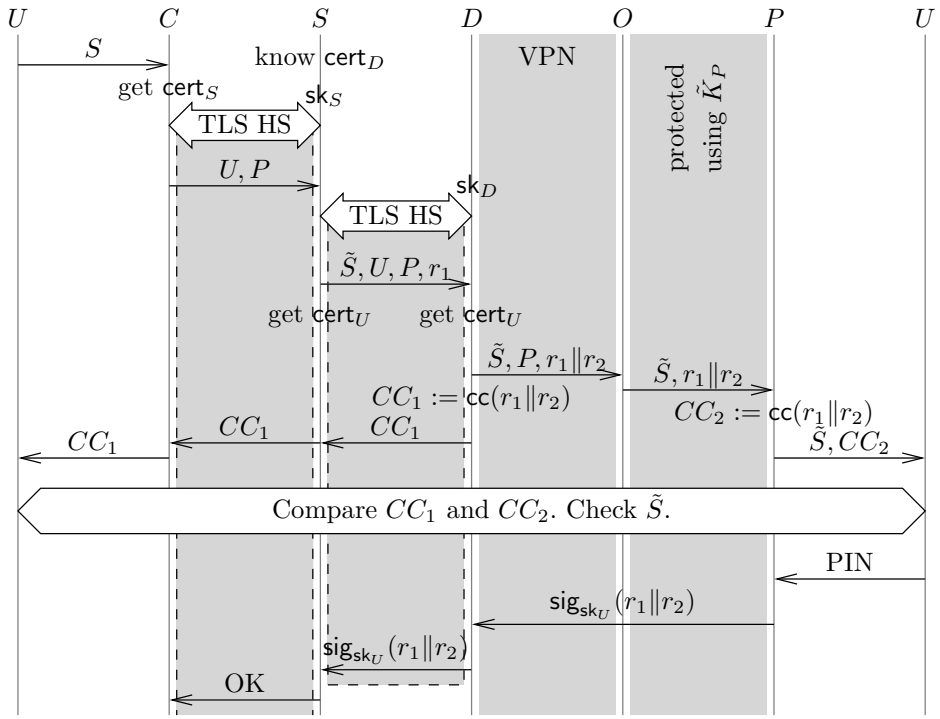

Fig. 1. Mobile-ID protocol

along with the name of the server S and the accompanying message m. The user checks that the control codes displayed by the client application and the phone are equal. The user also checks that the name of the server displayed by the phone is equal to the server he wanted to access, and the message m makes sense. If the checks succeed then the user instructs the phone that it is OK to sign the challenge, and provides the PIN for identification. The phone, receiving the PIN, signs the challenge $r_1 \| r_2$ and forwards it to D via O. DigiDocService D verifies that signature. If the signature verification is successful, and r_1 is not empty, then the signature is forwarded to S that also checks it, and upon success deems U to be authenticated. If r_1 is empty (i.e., S did not provide a challenge) then D does not forward the successfully verified signature to S, but only sends the confirmation that the verification succeeded. Again, S considers U to be authenticated. The TLS tunnel between C and S is then used for the regular communication.

2.1 Base security model

There are several entities, with several channels between them. Certain of those may be controlled by the adversary. In our "base" model we make the following assumptions about the security of various channels and entities:

1. There are several users and servers, some of them may be under adversarial control.
2. There is a single DigiDocService and mobile operator. They are honest. The channel between them is secure. In reality, these parties are relatively large organizations under significant public scrutiny. Still, we will relax this assumption in certain models.
3. The phones and client applications are under the control of their respective users.
4. The channel between a user and his client application is secure. So is the channel between a user and his phone. This is a reasonable assumption (for the base model, where we do not consider trojaned devices) as these channels are realized by the keyboards and screens of computers and phones.

The basic security property that we are interested in is the secrecy of the TLS keys agreed by honest clients and servers. We are also interested in correspondence properties: if a server S thinks that it talks to a client controlled by the user U using the key K and U is honest, then U must also think that it talks to the server S in a session where his client application C is using the key K (*integrity for servers*). Similar property must hold if we swap the user and the server (*integrity for clients*). Note that integrity for clients is derived directly from the properties of TLS because the server is authenticated during TLS handshake. TLS is a thoroughly researched protocol [12] and we know that it provides integrity for clients. Therefore we will subsequently only be concerned with the integrity for servers.

3 Formalization in ProVerif

ProVerif [7] is a protocol analyzer in the formal (or: Dolev-Yao) model [11]. To apply it to a protocol, it has to be formalized in a language reminiscent to the applied π-calculus [3]. In this calculus, messages are represented by *formal expressions* made up of free names and expression constructors, the set of constructors is fixed for a protocol. The process is expressed in a language containing primitives for sending and receiving messages (the channel has to be specified, too; it is a name), generating new names (modeling the generation of new keys, nonces, etc.), constructing and deconstructing messages, branching, sequential and parallel composition, and replication. The input language of ProVerif also contains means to specify the security properties (both secrecy and correspondence properties, as well as various process equivalences that we are not using here). ProVerif is a mature tool, having been used to check the security of various key-exchange [7, 2], authentication [8], fair exchange [1], secure storage [9], electronic voting [17, 5], etc. protocols. The tool is capable of modeling different cryptographic primitives, including Diffie-Hellman key exchange [2] and non-interactive zero-knowledge [6].

Our model of the Mobile-ID protocol, following the base security model consists of the following parts.

TLS handshake We follow the modeling by Tankink and Vullers [22]. They have verified that the TLS handshake provides integrity for clients. The TLS handshake is used as a subprotocol in two different places of the Mobile-ID protocol. We use the trick described by Haack [14] to include TLS handshake as a subprotocol, without duplicating its code.

Certification Instead of including a full-fledged CA process in our model, giving signatures to certificate requests, we have included a private expression constructor cert, such that $\mathsf{cert}(X, \mathsf{pk}_X)$ represents that pk_X is the public key of X. The privacy of the constructor means that the adversary cannot use it to construct new expressions. On the other hand, we have included destructors that the adversary can use and read both components of a cert-message. The honest users, servers and DigiDocService generate their keys and publish the corresponding certificates at the beginning of their processes. To give certificates to dishonest users and servers, we add a (replicated) process that takes a public key pk as an input, generates a new name n and outputs $\mathsf{cert}(n, pk)$ on a public channel. It is important that the name is newly generated, otherwise the adversary could issue new certificates to honest parties.

Phone registration The binding of the key $\tilde{K}_P$ to the phone P is handled similarly — there is a private binary constructor $\mathsf{phonereg}$ representing the binding of a key to a user's phone. There are also destructors to read both components of a $\mathsf{phonereg}$-message, but only the one giving the name of the user is public (i.e., can be invoked by the adversary). Binding a key to the phone of a dishonest user is handled similarly to certification. Actually, the process described in the previous paragraph is extended to also output a $\mathsf{phonereg}$-message.

User, client application and phone We model these parties as a single process (with several replicated subprocesses). The process first generates a new name and keys for signing and mobile communication and publishes the certificates for them. The process will then split into several parallel subprocesses, each of them replicated. These subprocesses are described below.

One of the subprocesses models the client application in one protocol session. It receives the name of the server to connect to (from the user subprocess), runs the TLS handshake with the server, verifying server's identity in that process, receives the control code from the server through the established TLS-tunnel and sends it to the user subprocess. The channel between the user and client subprocesses is a secure one; its name is generated before the parallel subprocesses start.

Another subprocess models both the user and the phone during one protocol session. It tells the client application to start connecting to a server (the name of the server is received from the adversary), gets back the control code from it, and also gets the challenge to be signed and information identifying the server from the network, encrypted with the key for mobile communication. The process

verifies whether the control code from the client application matches the control code of the challenge (also checks the identity of the server). If the check succeeds, it signs the challenge and sends it back, encrypted.

Third subprocess is used to indicate that this user is honest. It sends the name of the user on a private channel (a free name that the adversary does not have access to).

Other parties The processes modeling a server, the DigiDocService, and the mobile operator are straightforward. The server process first generates the name and the key of the server, publishes the certificate cert_S and then runs an unbounded number of processes implementing the server part of the Mobile-ID protocol. The name of the DigiDocService is globally known, hence the DigiDocService process starts by generating only the key and publishing the certificate for it. We use a private channel (a free name that the adversary cannot use) to model the VPN used for communication between the DigiDocService and mobile operator.

The whole system The analyzed process consists of the parallel composition of the client process (replicated), server process (replicated), DigiDocService process, mobile operator process, processes for TLS handshake (replicated) and the process for issuing certificates for dishonest clients and servers.

Checking the control code The control code consists of four decimal digits, hence collisions are easy to construct. It would be wrong to model the control code just by a message constructor with no additional equations as that would hide the very real possibility of control code collisions. In our model, we still introduce the constructor cc, such that $\mathsf{cc}(r)$ is the control code corresponding to r, but instead of using equality of terms to check the control code in the user process, we have introduced a binary predicate $\mathsf{TestCCEq}$ (ProVerif supports such introduction of predicates). The invocation of $\mathsf{TestCCEq}(r, c)$ is supposed to return true if c is the control code corresponding to r (recall that the user process receives the control code from the client process and the challenge from the mobile operator). Our model contains the clause $\mathsf{TestCCEq}(x, \mathsf{cc}(x))$. Additionally, it contains the clauses for modeling that given c, the adversary can construct messages of certain shapes whose control code is c. The shapes of these clauses depend on the attacks that the adversary may want to perform. In the weakest case (the adversary can find preimages of a given control code, but cannot control the shape of the preimage) the clause is $\mathsf{TestCCEq}(\mathsf{invcc}(x), x)$, where invcc is a new message constructor. We consider stronger cases when we study different security models. Our model does not consider the possibility that two control codes might be equal by chance. An honest DigiDocService can easily ensure that challenges with equal control codes are not awaiting signatures of the same user at the same time.

Security properties We are interested in two security properties — the secrecy of the keys of the TLS-tunnel between honest clients and servers, and the authentication of users to servers. Our model in ProVerif contains queries for verifying these two properties. For verifying the secrecy of keys, we have introduced a private free name M. The server process encrypts M with the keys of the TLS tunnel at the end of each protocol round (at the bottom of Fig. 1) and makes the resulting ciphertext public. The query asks ProVerif whether M is still secret.

For the correspondence properties we use the *events*. An event E is a program statement that is semantically equivalent to a no-operation, but records that the program point containing event E has been passed (E has happened). ProVerif can answer queries of the form "if event E_2 has happened, then must the event E_1 also have happened?" In our model, we add an event $\mathsf{ServerEnd}(U, S, k)$, where k is the key for the TLS-tunnel between S and C, to the end of the server process, after it has accepted that user U has been authenticated. We also add an event $\mathsf{UserEnd}(U, S, k)$ at the point where the user has completed all of his steps to be accepted by the server — at the point where the user must enter his PIN to the phone. The user process does not normally have the key k. Therefore the client process will send k to the user process, too. The query asks whether the event $\mathsf{ServerEnd}(U, S, k)$ implies $\mathsf{UserEnd}(U, S, k)$.

Both properties are easily invalidated if the user is dishonest. Hence the server performs the actions for both properties (publishing the encryption of M, and performing the event $\mathsf{ServerEnd}$) only if the user is honest. The user is honest if the server can receive his name over the private channel for honest user names (see the description of user, client application and phone processes above).

4 Verification results

The Mobile-ID protocol, as we have modeled it in Sec. 3, following the security model of Sec. 2.1 is deemed secure by ProVerif — the correspondence property holds and the message M cannot be found by the attacker. Still, this only reflects an in some sense "ideal" situation. Let us now consider the protocol where certain things go wrong with respect to the base security model.

4.1 Attacker controls DigiDocService

DigiDocService is a mediator of messages, helping the protocol to proceed. It would be unnatural if the *security* of the protocol depended on its actions. It is straightforward to model DigiDocService being under adversarial control — we make public its secret key sk_D, as well as the channel between it and the mobile operator.

Being under adversarial control, the DigiDocService is expected to look for collisions in control codes for challenges. As it can fix the second half of the challenge, we expect that DigiDocService desires to solve the problems of the following form: given c and r_1, find r_2 so that $\mathsf{cc}(r_1 \| r_2) = c$. This is a reasonably

solvable problem, and we add a clause to the definition of $\mathsf{TestCCEq}$ stating its solvability. Namely, we introduce a binary message constructor postc and state that $\mathsf{TestCCEq}((r_1 \| \mathsf{postc}(c, r_1)), c)$ holds.

ProVerif finds an attack violating both security properties. This attack should not even be so surprising, because the construction of the signature $\mathsf{sig}_{\mathsf{sk}_U}(r_1 \| r_2)$ violates certain prudency criteria for the construction of cryptographic protocols [18, Ch. 11]. If the adversary wants to masquerade as U to a server S then it waits until U wants to contact some server S', and proceeds as follows:

- A contacts S and performs the TLS handshake with it. At the same time, U is performing the TLS handshake with S'.
- A identifies itself to S as U. S contacts DigiDocService D (under control of A), performs the TLS handshake with it and forwards it the name U (and P) together with its own name $\tilde{S}$ (including the additional message m) and its half of the challenge r_1. At the same time, U identifies itself to S', which also performs the TLS handshake with D and forwards to it the names U and P, its own name $\tilde{S}'$ and the half of the challenge r_1'.
- The adversary (as D) constructs r_2 and r_2' so, that $\mathsf{cc}(r_1 \| r_2) = \mathsf{cc}(r_1' \| r_2')$. Let c be this control code. The adversary (as D) sends c back to S and S'. It also sends $\tilde{S}'$, P, and $r_1 \| r_2$ to the mobile operator, which forwards them to P.
- The server S' sends c back to U. The phone P shows $\tilde{S}'$ to the user U. The user agrees that it intended to contact S'. The phone also shows the control code of $r_1 \| r_2$ to the user. This happens to equal c.
- The user enters his PIN to the phone and the phone signs $r_1 \| r_2$. This signature is forwarded to S (via the mobile operator and the adversary posing as DigiDocService), and S accepts the connection with A as coming from U.

Note that here the adversary only controls D, and not any other parties. Therefore this is a very powerful attack.

The attack succeeds even if the collisions for control codes were impossible to construct. Impossibility of collisions means that the control codes must be so much longer (at least 40-50 decimal places) that it would seriously degrade the usability of the system. In this case, the attack is possible if S' is under adversarial control. Compared to the described attack, we now just take $r_1' = r_1$ and $r_2' = r_2$, and we do not have to look for collisions.

A prudent protocol design guideline says that when constructing a signature, let the name of the intended verifier be a part of the signed message. This guideline has not been followed in the design of the Mobile-ID protocol. We could modify the protocol by letting P include the name S under the signature it generates, and subsequently verifying that a correct name has been included.

This change still does not fix the protocol, but now the original attack succeeds only if $S = S'$. In other words, the user U initiates one session with the server S and the attacker initiates a different session at the same time. The adversary (in the role of D) again finds r_2 and r_2' so that there is a collision in the control codes. The modified protocol might be secure if a server does not allow

the same alleged user to run two sessions in parallel. Unfortunately, we do not know of a simple means to model such restriction in ProVerif.

ProVerif deems the protocol secure if both modifications (no control code collisions and server name under signature) are made. To ensure that adversarially controlled DigiDocService does not generate control code collisions, the server itself should generate the whole challenge. In terms of Fig. 1, r_2 should be the empty string and r_1 should be long enough to be unpredictable. The server must also make sure to not issue challenges with the same control code for parallel sessions of allegedly the same user. It goes without saying that the control code CC_1 sent to the user via his client application must be computed by the server, not the DigiDocService.

4.2 Attacker partially controls the client

One of the goals of the Mobile-ID protocol was to reduce the effect that a compromised client machine might have on the security of authentication. Clearly, if the adversary has completely taken over the client computer, then it knows the keys for the TLS-tunnel between the client and the server and can listen and speak on behalf of the user. Still, even in this case the adversary cannot contact a server S on behalf of a user U while the user U remains completely passive: ProVerif claims that even in this case the event $\mathsf{ServerEnd}(U, S)$ implies the event $\mathsf{UserEnd}(U, S)$ (note that we do not include the key for the TLS-tunnel in the arguments of these events) and moreover, the correspondence is injective: for each action of the user (instructing the phone to sign a challenge), the adversary can start at most one session with the server. To model in ProVerif that the adversary controls the client machine, we make public the secret that this process uses: the channel between the client application and the user.

A keylogger does not have to take over the whole machine in order to cause harm (record the PIN of the ID card). A similarly interesting case for the Mobile-ID protocol is, when the malware has not taken over the whole machine, but can influence how the control code is shown to the user. This case models malware that can redraw the screen. This change is simple to model — in Fig. 1, the value CC_1 is received by the user U not from C, but from public network.

ProVerif finds, that if the adversary controls the value of CC_1 as presented to U, then the protocol is insecure. If the adversary A wants to masquerade as the user U to a server S, then it proceeds as follows.

- Wait until U himself contacts S. As we explained in the first paragraph, it is impossible to initiate a session (as U) with S, unless U himself also wants to contact S.
- Start a session with S, claiming to be U. Let both sessions proceed to the point where DigiDocService has constructed control codes c (for U) and c' (for A) and sent them to S and to P.
- The adversary makes sure that the challenge with the control code c' reaches P first. This can be achieved with right timing. The adversary also makes sure that the second control code is not shown to the user before it has

accepted c'. By our experimentations with DigiDocService[3], this condition trivially holds — a mobile phone does not hint of the existence of a second incoming control code before the user has taken action on the first one.
– The adversary receives c', the client application receives c. The dishonest client application now contacts the adversary, learns c', and shows it to the user instead of c. The user confirms that the client application and the phone show the same control code c' and instructs the phone to sign the challenge. This challenge corresponds to the session between A and S.

The attack should be avoidable if the server does not start several sessions with the same user in parallel. Indeed, if a session has ended and the user has generated the event $\mathsf{UserEnd}(U, S)$ then the server has also generated the event $\mathsf{ServerEnd}(U, S)$ and because of injective correspondence, this event $\mathsf{UserEnd}(U, S)$ cannot be used to match a different $\mathsf{ServerEnd}(U, S)$ taking place later (presumably because of adversarial activity).

4.3 User confused regarding the server names

An important class of attacks are *semantic attacks* where the adversary tries to convince the user that a wrong statement holds. One example of such attacks are the phishing web-sites masquerading as legitimate ones. They typically have names similar to the one they are trying to masquerade. Authentication using an ID card is relatively immune to such attacks — while an attacker can obviously make the user connect to a fake server (if the user does not notice its fakeness), this cannot be used to masquerade the user to a legitimate server.

We studied how well the Mobile-ID protocol fared against such attacks. We assumed that there is an adversarially controlled server S' that is hard to distinguish from a legitimate server S. It turned out, that a classical man-in-the-middle attack is possible, allowing the adversary to masquerade as U to S. In this attack, U connects to S' thinking it is S, while the adversary (masquerading as U) connects to S. The server S contacts D and the phone of the user receives the challenge and the information $\tilde{S}$ identifying the server. The control code is also sent from D back to S, which forwards it to A, which forwards it to U as S'. The phone shows S as the name of the server, the user is connected to S', but we assume that he does not notice the difference. The control codes shown by the phone and the client application are the same. The user hence tells the phone to sign the challenge and S will accept A as U. The attack works even if the name of the server is included in the signed message.

4.4 Server chooses the control code

When server S contacts the DigiDocService D, it sends it not only the name S, but also the up to 40 characters long message m; both S and m will be shown to the user on his phone. A typical picture of the phone screen is shown in Fig. 2a.

[3] http://www.sk.ee/DigiDocService/DigiDocService_2_3.wsdl

Here S equals "TheBank" and m equals "Enter?". The next lines have been produced by software running inside the phone (actually, inside the SIM-card). They inform the user that the control code of the challenge is 1234.

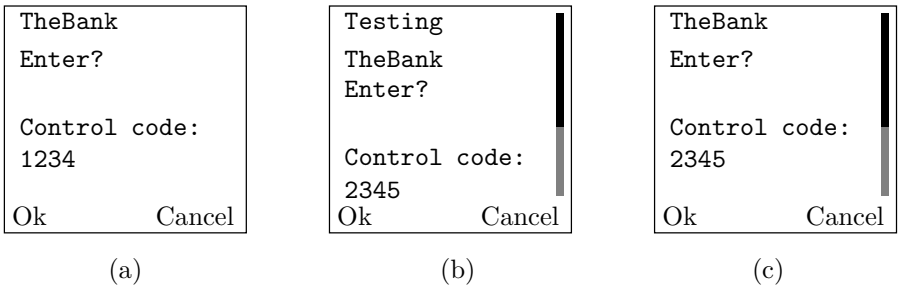

Fig. 2. Typical screen for comparing the control code

The possible values for S have been enumerated by the DigiDocService D and D also weakly authenticates S by its IP-address. The server named "Testing" may connect from any IP-address. We have experimented with DigiDocService and various values of m, using this server identity. We have discovered that if m contains embedded newlines, then these are shown as line breaks on the phone screen. E.g., if m equals "TheBank\nEnter?\n\nControl code:\n2345\n" then the phone screen (for certain models) will look as shown in Fig. 2b. Here the entire message shown by the phone has not fit on the screen and a scroll bar has appeared on the right-hand side. If we scroll down, we see the control code that has been computed by the phone itself and may be different from 2345. Depending on the model of the phone, this scroll bar may be rather hard to notice.

We believe that with IP-spoofing, we can cause the picture in Fig. 2c to appear on the phone screen. Also, if the adversary controls either TheBank or the DigiDocService (as we have argued before, this case has to be analyzed, too) then it is straightforward to make this picture appear, as the DigiDocService can control which S and m are sent to the phone. Hence we conclude that a malicious server or DigiDocService can (under certain circumstances) control which four-digit number is shown to the user as the control code.

We have modeled this scenario with ProVerif. The necessary modifications involve several parts of the model, as the fake control code is inserted at the server, and then travels to DigiDocService, mobile operator, phone, and the user. At the same time, the changes are rather straightforward.

We have considered the case where the DigiDocService is honest, but a malicious server is capable of entering a fake control code, eventually shown to the user by the phone. Somewhat surprisingly, the protocol is still deemed secure. One may conjecture that the user might not need to perform the equality check of control codes at all. Of course, this is not so; there exists a straightforward

parallel attack: both U and A connect to S, both claim to be U, the challenge for A's session is the first to arrive at U's phone, U does not check the equality of control codes and causes this challenge to be signed. Note that U still checks that the phone shows the name of the server S (the opposite is considered in Sec. 4.3). The reason why there is no attack if a malicious server can pick a fake control code, is that to use this capability, the attacker has to set up a malicious server S' whose name will be shown to and rejected by the user.

In Sec. 4.1 we gave some suggestions to handle a dishonest DigiDocService. We have checked the security of our modifications (server name under signature, challenge generated entirely by the server, no control code collisions) if a server or the DigiDocService can also choose the control code that the phone shows. ProVerif gives us an attack. The attack is similar to the attack presented in Sec. 4.2. The only difference is that now the attacker changes the control code shown by the phone, not the control code shown by the client application. Again, the attack should not work if the server does not start several sessions with the same user in parallel.

5 Proposed improvements

We suggest the following modifications to the Mobile-ID protocol to increase its security:

- When the phone signs the challenge $r_1 \| r_2$ for the server S, the signed message should not be $\mathsf{sig}_{\mathsf{sk}_U}(r_1 \| r_2)$, but $\mathsf{sig}_{\mathsf{sk}_U}(r_1 \| r_2, S)$. The presence of S under the signature must be checked by parties receiving that signature.
- r_2 should be a constant, most naturally the empty string. The control code CC_1 should be computed by S, not D. The server S should generate the challenges r_1 in such a way that the sessions of the same alleged user U have challenges with different control codes.

We also suggest that the user interface of the phone should be modified in a way that the control code is always in the same place at the phone screen, and always visible when the message is first shown to the user. This can be achieved by showing the control code before the message m, or by appropriately filtering m. Users should also be educated to look for the control code in a certain place.

6 Summary

Above, we have considered attackers of various strength. They all had the ability to initate protocol sessions; they controlled certain users and servers and had no access to the phones of the users. Their strength varied along the following dimensions:

- $\mathbf{d}_1$ — control over the DigiDocService or mobile operator (possible values: 0 — no control, 1 — full control);

– $\mathbf{d}_2$ — control over the client application (0 — no control, 1 — can change displayed CC, 2 — full control);
– $\mathbf{d}_3$ — ability to confuse the user about server names (0 — no, 1 — yes);
– $\mathbf{d}_4$ — ability for a compromised server to pick the control code shown on phone screen (0 — no, 1 — yes).

We see that the abilities of attackers may include the corruption of any party in Fig. 1, except the phone P, taking the advantage of the user interface issues in both C and P, and phishing attacks. We have not considered the attacker gaining significant control over the phone. Indeed, any reasonable attack model would allow the adversary to learn the PIN entered from the keypad of the phone; the knowledge of the PIN gives the adversary full capabilities of masquerading as the user U. We have also not considered the user's failure to compare the control codes shown by the client application and the phone, but this is subsumed by the dimension $\mathbf{d}_2$. To summarize, we believe that we have not left any significant attack vectors without consideration.

We have proposed two mutually independent protocol modifications. These propositions introduce dimensions on whether they have been taken into account.

– $\mathbf{d}_5$ — is the name of the server included under the signature of the challenge? (0 — yes, 1 — no)
– $\mathbf{d}_6$ — is the half r_2 of the challenge empty? (0 — yes, 1 — no)

Another dimension is introduced by an honest server's behaviour when allegedly the same user attempts to authenticate himself several times in parallel:

– $\mathbf{d}_7$ — does S allow parallel sessions with the same U? (0 — no, 1 — yes, but picks challenges with different control codes, 2 — yes) Note that $\mathbf{d}_7 = 0$ means that the server lets a session with a user U to time out before agreeing to participate in a different session with U. This may make denial-of-service attacks too simple.

Let $\mathbf{L}_j^i$ denote the predicate $\mathbf{d}_j \leq i$. Our analysis shows that the security properties described in the end of Sec. 3 hold if

$$(\mathbf{L}_1^0 \vee (\mathbf{L}_5^0 \wedge \mathbf{L}_6^0 \wedge \mathbf{L}_7^1)) \wedge \mathbf{L}_2^1 \wedge (\mathbf{L}_2^0 \vee \mathbf{L}_7^0) \wedge \mathbf{L}_3^0 \wedge (\mathbf{L}_4^0 \vee \mathbf{L}_1^0 \vee (\mathbf{L}_5^0 \wedge \mathbf{L}_6^0 \wedge \mathbf{L}_7^0)) \ .$$

Indeed, the justification for each of the conjuncts is the following:

– We showed in Sec. 4.1 that an adversarially controlled DigiDocService is capable of breaking the protocol unless the modifications stated in Sec. 5 were introduced.
– If the adversary has full control over the client application, then it can take over the connection between C and S.
– In Sec. 4.2 we showed that if the adversary can change the control code shown to the user by the client application, then there exist an attack that requires parallel sessions with the same server.
– In Sec. 4.3 we argued that the mobile-ID protocol does not protect against phishing attacks, even if we implement the modifications in Sec. 5.

– In Sec. 4.4 we showed that the capability for a malicious server to choose the control code displayed by the phone is not enough for breaking the security properties, but if the DigiDocService is also under adversarial control then the modifications of Sec. 5 no longer suffice to preserve the security, but parallel sessions between the same alleged user and server must be ruled out. Hence we suggested in Sec. 5 to make sure that $\mathbf{L}_4^0$ holds.

7 Conclusions

We have analyzed the security of the Mobile-ID protocol introduced by an Estonian CA and Estonian and Lithuanian mobile operators. We have discovered some weaknesses in the protocol which manifest under strong adversarial models. Despite these weaknesses, we believe that the usage of the protocol can continue in the immediate future. Indeed, we believe that the attack vectors included in those adversarial models either will not materialize in the immediate future, or their materialization would allow attacks of similar success against other authentication methods, sometimes including ID card based methods. Still, the weaknesses should nevertheless be fixed with high priority.

Our analysis also shows that compared to other methods of authentication (passwords, one-time passwords, PIN-calculators), Mobile-ID does not offer significant protection against user errors or weaknesses of the client application. We conclude that it is too premature to state that *modulo* negligible risks, Mobile-ID is at least as secure as authentication with ID card [21].

Acknowledgments

This research has been supported by Estonian Science Foundation, grant #6944, by the European Regional Development Fund through the Estonian Center of Excellence in Computer Science, EXCS, and by Sampo Pank. We are grateful to Dan Bogdanov, Ilja Livenson, and Mari Seeba for fruitful discussions.

References

1. M. Abadi, B. Blanchet. Computer-Assisted Verification of a Protocol for Certified Email. In Static Analysis, 10th International Symposium (SAS'03), LNCS 2694, pages 316–335, San Diego, California, June 2003.
2. M. Abadi, B. Blanchet, C. Fournet. Just Fast Keying in the Pi Calculus. In Programming Languages and Systems: Proceedings of the 13th European Symposium on Programming (ESOP'04), LNCS 2986, pages 340-354, Barcelona, Spain, March 2004.
3. M. Abadi, C. Fournet. Mobile values, new names, and secure communication. In 28th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL), pages 104–115, London, UK, January 2001.
4. AS Sertifitseerimiskeskus. DigiDocService specification, v. 2.122, April 24th, 2007. http://www.sk.ee/files/DigiDocService_spec_eng.pdf

5. M. Backes, C. Hritcu, M. Maffei. Automated Verification of Remote Electronic Voting Protocols in the Applied Pi-Calculus. In 21st IEEE Computer Security Foundations Symposium, CSF 2008, Pittsburgh, Pennsylvania, pages 195–209, June 2008.
6. M. Backes, M. Maffei, D. Unruh. Zero-Knowledge in the Applied Pi-calculus and Automated Verification of the Direct Anonymous Attestation Protocol. In 2008 IEEE Symposium on Security and Privacy, pages 202–215, May 2008.
7. B. Blanchet. An Efficient Cryptographic Protocol Verifier Based on Prolog Rules. In 14th IEEE Computer Security Foundations Workshop (CSFW-14), pages 82–96, Cape Breton, Nova Scotia, Canada, June 2001.
8. B. Blanchet. From Secrecy to Authenticity in Security Protocols. In 9th International Static Analysis Symposium (SAS'02), LNCS 2477, pages 342–359, Madrid, Spain, September 2002.
9. B. Blanchet, A. Chaudhuri. Automated Formal Analysis of a Protocol for Secure File Sharing on Untrusted Storage. In IEEE Symposium on Security and Privacy, pages 417–431, Oakland, CA, May 2008.
10. T. Dierks, E. Rescorla. The Transport Layer Security (TLS) Protocol, Version 1.1. IETF Network Working Group, RFC 4346, April 2006.
11. D. Dolev, A. C. Yao. On the security of public key protocols. IEEE Transactions on Information Theory 29(2):198–207, 1983.
12. S. Gajek, M. Manulis, O. Pereira, A.-R. Sadeghi, J. Schwenk. Universally Composable Security Analysis of TLS. In 2nd International Conference on Provable Security, ProvSec 2008, LNCS 5324, pages 313–327, Shanghai, China, October 2008.
13. S. Golovanov, A. Gostev, D. Maslennikov. Kaspersky Security Bulletin 2008: Malware Evolution January - June 2008. http://www.viruslist.com/en/analysis?pubid=204792034#9
14. C. Haack. Verification of Security Protocols, ProVerif's Resolution Method, lecture slides. March 2008. http://www.cs.ru.nl/~chaack/teaching/2IF02-Spring08/
15. idBlog. EMT Launches the Mobiil-ID Service. http://www.id.ee/blog_en/?p=20, May 2nd, 2007.
16. ID.ee. Mobile-ID main page. http://www.id.ee/10995, November 20th, 2008.
17. S. Kremer, M. Ryan. Analysis of an Electronic Voting Protocol in the Applied Pi Calculus. In Programming Languages and Systems, 14th European Symposium on Programming, ESOP 2005, LNCS 3444, pages 186–200, April 2005.
18. W. Mao. Modern Cryptography: Theory and Practice. Prentice Hall, 2003.
19. M. Myers, R. Ankney, A. Malpani, S. Galperin, C. Adams. X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP. IETF Network Working Group, RFC 2560, June 1999.
20. R. Šablinskas. Summary of Mobile-ID launch in Lithuania. Minutes of the Baltic WPKI Forum Steering Committee, October 31, 2007. http://wpki.eu/Launch-of-mobile-ES-BalticWPKI.pdf
21. Security Analysis of Mobile ID (Summary, in Estonian). Ordered by Department of State Information Systems, fulfilled by Jaak Tepandi. July 11th, 2008. http://www.riso.ee/et/files/MOBIIL-ID_kokkuvote_11-07-2008.pdf
22. Carst Tankink, Pim Vullers. Verification of the TLS Handshake protocol. May 20th, 2008. http://www.cs.ru.nl/~chaack/teaching/2IF02-Spring08/tv-report.pdf