Anders Moen Hagalisletto · Arne Riiber

Using the mobile phone in two-factor authentication

the date of receipt and acceptance should be inserted later

Abstract In this paper we present a commercial protocol, developed by a norwegian start-up copmany, for using a mobile terminal as a password calculator that could potentially be used towards any service provider on the internet. We report our experiences by specification, validation, and analysis of the protocol, in particular the threat of phishing attacks is investigated.

1 Introduction

Since there is more than mobile phone per citizen in Europe, it has a potential for interacting with services, anywhere, anytime, and thus fits into the cornerstone of Mark Weiser's vision of ubiquitous computing. The Key Server application described in this paper, allows the mobile phone to be used as a one time password (OTP) calculator. It can be used for authentication to services. In the paper we shall investigate the commercial protocol, and analyze how the protocol (or more accurately a couple of protocols) resist various standard as well as phishing attacks.[1] The protocol is used in an application that currently is in the pilot stage, that is, restricted to a limited number of users. It has not reached the stage of volume production yet.

The threat of traditional attacks [5] and phishing attacks ([2], [3], [1]), pose new challenges to the design and analysis of security protocols. The estimated cost of the current phishing attacks in US was estimated to 1.2 billion dollars in 2004, and Gartner has estimated that phishing attacks cost the US something in the region of $ 2.8 billion last year. The average individual loss per attack has risen from $256 in 2005 to a staggering $1244 in 2006. Phishing attacks are considered to be "social" attacks, and have therefore not achieved the attention from the formal methods community as the severity of the phenomenon deserves. For the designers of security protocols and e-commerce applications, it is therefore important to understand what kind of attacks that can be mounted in order to set up appropriate defenses. In particular it is crucial to understand what effects a phishing attack has on the surrounding protocol events.

Contrary to common beliefs in computer security, phishing attacks can be given a precise interpretation: An agent is fooled to trust a malicious site on the Web, and reveals sensitive information through a login event. Many e-business applications including banks use the Secure Socket Layer protocols (SSL) or it successor Transport Layer Security protocol (TLS). Both protocols have been subject to formal analysis ([8], [9], and [10]), and no crucial deficiencies in the protocols have been found. The protocols SSL/TLS are commonly regarded as secure. Typically a SSL/TLS connection involve a one-way authentication session from the client to the server. Two-way authentication is rather not often used, since it requires extensive book-keeping of certificates on a variety of clients. Hence in practice many phishing attacks can be described as a man-in-the-middle attack where the good agent establishes a corrupt SSL/TLS connection to the intruder. The intruder then establishes a SSL/TLS connection with the actual receiver, such that the receiver use this latter connection to send messages back to the intended client.

In the key server application all requests initiated from the client and service provider and their corresponding responses are wrapped inside TLS/SSL.

The method used in the paper can be separated into three steps that are performed iteratively:

(a) *formal specification*,
(b) *static analysis*,
(c) and *dynamic analysis*.

Anders Moen Hagalisletto
Birkeland Innovation and Norwegian Computing Center
Gaustadalleen 23, N-0371 Oslo
E-mail: andersmo@ifi.uio.no

Arne Riiber
enCap, Research Park
Gaustadalleen 21, N-0371 Oslo
E-mail: arne.riiber@encap.no

[1] The basic authentication mechanism is covered by the patent application document (PCT WO/2007/039806).

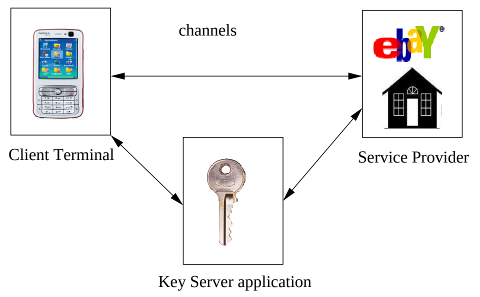

Fig. 1 A standard scenario with a client, a key server, and a service provider.

Each iteration involves an sequence consisting of the steps (a - c). Initially, an informal description of the protocol is translated into a precise specification in the formal language for protocols $\mathscr{L}_P$ [6]. Then a static *analysis* of the formal specification is performed. Our notion of static analysis is two-fold: First, the specification is refined in an automated way. The refined protocol contains information about all the local assumptions of every agent participating in the protocol. The refined protocol gives an explicit explanation of the intended execution of the protocol, by including assertions about possessed keys, timestamps, nonces, encryption and decryption. Assumptions also include *local actions*: These include explicit construction of fresh nonces, keys, timestamps and composite data like composite keys and encryptions. Our approach is closer to protocol execution than the typical logical approach: the belief operator is interpreted *operationally.*

Second, a refined protocol might be *validated.* This means that the correctness of every assertion, i.e. every belief in the refined protocol is checked. An assertion is considered valid if it is obtained in a legal way either by communication or by the cryptographic operations. A valid protocol can be subject to *dynamical* analysis. The PROSA tool [6] is equipped with an execution engine, capable of simulating both intended protocols and attack descriptions.

The application investigated in this paper contains explicit deletion of objects, both basic and composite cryptographic entities are deleted on the client. Typical appraoches, like BAN logics [4], do not handle removal of beliefs. An operational interpretation of the belief operator proposed in [6] can model deletion by the following construct: $\mathsf{Enforce}_a(\neg\mathsf{Bel}_a(\varphi))$ reads "a is enforced to not believe the sentence φ", or "a removes φ". Informally this means that agent a removes the occurrence of the belief φ. Removal of cryptographic entities is important on the mobile device, in order to account for scenarios where a malicious agents steals the terminal and tries to access confidential information.

The application analyzed in this paper is characterized by three medium size protocols. It is well known that deploying reachability analysis to protocols that are not small, is a hard problem which quickly runs into state space explosion. It is therefore important to develop "softer techniques" that can be used to give weaker but computationally feasible analysis to commercial protocols.

The investigation revealed a large collection of potential attacks on the Key Server application: attacks on both confidentiality and authenticity. The initial specification did not contain any detailed specification of SSL/TLS wrapping of the messages. This resulted in a large number of attacks on the registration and authentication protocol: we described totally 85 attacks. The attacks involved impersonation of each of the three agents in question, the client, the key server and the service provider. The protocols were then revised with SSL/TLS wrappings. The SSL/TLS encapsulation is assumed to be secure, and hence only a couple of phishing attacks could be launched.

2 An authentication framework for mobile terminals

The idea behind the Key Server application is to utilize a mobile terminal as a one-time password calculator. The application is an answer to the following two questions:

(i) What do the banks really need?
(ii) Can we utilize something the user already has?

Banks need strong authentication, in other words, what the user has, and what the user knows. It is also possible to utilize the mobile phone as the proof of possession element, since mobile phones are frequently used. In authentication services, a one time password calculator is referred to as a possession factor. Two-factor authentication solutions, used for online banking and other advanced e-services, comprises of both a possession factor and a knowledge factor. A knowledge factor is a user secret like a password or a PIN code.

The Key Server application is intended to enable service providers to use a mobile terminal as a possession factor in single- or two-factor authentication solutions. The potential benefit of the Key Server application is cost effectiveness, user friendliness and secure authentication. Combining the mobile terminal as a possession factor with the user's knowledge factor, enables service providers to utilize mobile terminals in different implementations of secure two-factor authentication. In Figure 2, a typical scenario is shown: there is a large number of clients running the application on various mobile platforms, only one key server, and a limited number of service providers.

The Key Server application basically consists of three protocols: a download, registration, and authentication protocol. In this section we describe how these protocols work.

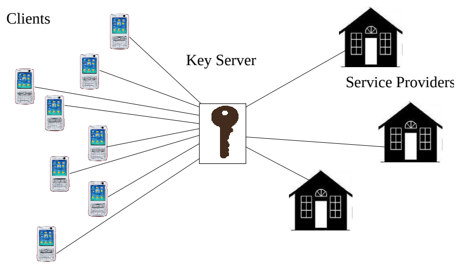

Fig. 2 Relations between the clients, a key server, and some service providers.

2.1 The specification of the protocols

We use the standard notation for the specification of authentication protocols, where A and B denote agents, and M denote a message content:

$$A \longrightarrow B \quad : \quad M$$

The message content might itself contain agent names and other basic cryptographic entities like nonces N text-strings T, and keys K. In addition there are composition operators like concatenation denoted by ',' (comma), encryption operator $E(K : M)$ and hashing $\mathsf{H}[M]$, where K denotes a key and M is some message content. The encryption function used is 128-bits Rijndael also denoted Advanced Encryption Standard (AES) and the hashing algorithm applied is SHA-1.

2.1.1 The download protocol

The purpose of the download protocol is to safely install the succeeding protocols, the *registration protocol* and the *authentication protocol*.

$$\begin{array}{lllll}
(\mathrm{D}_1) & A \longrightarrow Q & : & T_{\text{phone}} \\
(\mathrm{D}_2) & Q \longrightarrow A & : & T_{\text{URL}}, U \\
(\mathrm{D}_3) & A \longrightarrow Q & : & U, T_{\text{capabilities}} \\
(\mathrm{D}_4) & Q \longrightarrow A & : & T_{\text{verification}} \\
(\mathrm{D}_5) & A \longrightarrow U & : & T_{\text{request}} \\
(\mathrm{D}_6) & U \longrightarrow A & : & \mathbf{Reg}, \mathbf{Auth}
\end{array}$$

In the first message (D_1), the user's phone number T_{phone} is sent to the key server Q as request to download the key server application. Then Q replies with the URL for the code to be downloaded in the message (D_2). The URL is modeled as a distinct agent U, that could be thought of as a sub-agent of Q. The user confirms to open the URL, the mobile phone contacts the key server Q, and the mobile phone exhibits the terminal capabilities $T_{\text{capabilities}}$. If the capabilities of the terminal is sufficient for running the application, the key server replies (D_4) with a message expressing that the server has verified the terminal capabilities. The user then clicks on the URL (D_5) in order to download the application, in other words the protocols for registration **Reg** and authentication **Auth** in the final message (D_6).

2.1.2 The registration protocol

After having downloaded the registration protocol, the user can start to register. The user wants to register at a service provider that is connected to a key server Q, in order to subscribe or use a certain service. There are three agents involved: the client A, the Key Server server Q and an service prover S. The protocol contains several non-cryptographic entities, represented as text-strings: the service name requested by the service provider T_{service}, the client's phone number T_{phone}, the client's PIN code T_{pin}, and the client's unique mobile terminal number (so called IMEI[2]) T_{imei}. In addition the nonce N^S_{acode} denotes the activation code generated by the server, and N^A_{cref} denotes a nonce generated by the client, called the client-reference. There are two keys involved in the registration, the secret transaction key K_{AQ} and the non atomic key $\mathsf{H}[T_{\text{pin}}, T_{\text{imei}}, N^A_{\text{cref}}, S]$. The latter key is called the *security code.*

$$\begin{array}{lllll}
(\mathrm{R}_1) & A \longrightarrow S & : & T_{\text{service}}, T_{\text{phone}}, Q \\
(\mathrm{R}_2) & S \longrightarrow A & : & N^S_{\text{acode}} \\
(\mathrm{R}_3) & S \longrightarrow Q & : & A \\
(\mathrm{R}_4) & Q \longrightarrow A & : & K_{AQ} \\
(\mathrm{R}_5) & A \longrightarrow Q & : & \mathsf{H}[T_{\text{pin}}, T_{\text{imei}}, N^A_{\text{cref}}, S], \\
 & & & E(\mathsf{H}[T_{\text{pin}}, T_{\text{imei}}, N^A_{\text{cref}}, S] : N^S_{\text{acode}})
\end{array}$$

In the first message, a client A initiates a registration process at a particular service provider S, that offers a particular service T_{service}. The service provider replies with the activation code nonce N^S_{acode} sent to A. In the third message, S activates the key server Q, by telling Q that A is performing a registration. Then Q sends the first transaction key K_{AQ} to A. The client generates the nonce N^A_{cref}, chooses a PIN code, and then generates the security code: $\mathsf{H}[T_{\text{pin}}, T_{\text{imei}}, N^A_{\text{cref}}, S]$. Finally the client A use this code as a symmetric key to encrypt the activation code. Both the key and the cipher-text currently generated is returned to Q. The key server Q stores the security code for later authentication sessions.

At the end of the session both the PIN code and the security code are removed, in order to protect against the possibility that an illegitimate user (a thief) controls the terminal and tries to use the application for authentication by reconstructing the security code. For the same reason the client reference, a number that is generated for each registration session, is protected by the transaction key K_{AQ}, and then stored at the client's terminal for later use, as in clause (C_1) below. The tail of the registration protocol consists of a chain of *local* operations:

$$\begin{array}{ll}
(\mathrm{C}_1) & A \text{ performs } E(K_{AQ} : N^A_{\text{cref}}) \\
(\mathrm{C}_2) & A \text{ removes } K_{AQ} \\
(\mathrm{C}_3) & A \text{ removes } T_{\text{pin}} \\
(\mathrm{C}_4) & A \text{ removes } N^A_{\text{cref}} \\
(\mathrm{C}_5) & A \text{ removes } \mathsf{H}[T_{\text{pin}}, T_{\text{imei}}, N^A_{\text{cref}}, S]
\end{array}$$

The deletion of objects is performed in order to protect the device from tampering of the terminal itself. In particular: the PIN code is deleted and is intended to be

[2] IMEI stands for "International Mobile Equipment Identity". Some mobile phone vendors put restrictions on which software is allowed to retrieve the IMEI.

kept in the mind of the client's legitimate owner and the client reference is deleted after every session so that it is not possible for an intruder to reconstruct the security code.

2.1.3 The authentication protocol

In order to run a session of the authentication protocol, it is required that the client A has registered at a service provider. The atomic entities in the authentication protocol includes a text-string indicating an authentication request T_{auth} and three nonces: a *session identifier* N^Q_{sessid} and a *challenge* N^Q_{chall}, and *display OTP nonce* N^Q_{dotp}, each entity is generated by the key server Q. Two transaction keys are transmitted: the old key K_{AQ} and fresh key K'_{AQ}. In case the current authentication is the first, then K_{AQ} is identical with the transaction key obtained in the registration session, else it refers to the transaction key in the previous authentication session.

$$\begin{array}{llll}
(\mathrm{A}_1) & A \longrightarrow S & : & T_{\text{auth}} \\
(\mathrm{A}_2) & S \longrightarrow Q & : & A, S, T_{\text{phone}} \\
(\mathrm{A}_3) & Q \longrightarrow S & : & N^Q_{\text{sessid}} \\
(\mathrm{A}_4) & Q \longrightarrow A & : & N^Q_{\text{sessid}} \\
(\mathrm{A}_5) & A \longrightarrow Q & : & T_{\text{auth}}, N^Q_{\text{sessid}} \\
(\mathrm{A}_6) & Q \longrightarrow A & : & N^Q_{\text{chall}}, S, K_{AQ}, K'_{AQ} \\
(\mathrm{A}_7) & A \longrightarrow Q & : & E(\mathsf{H}[T_{\text{pin}}, T_{\text{imei}}, N^A_{\text{cref}}, S] \ : \ N^Q_{\text{chall}}) \\
(\mathrm{A}_8) & Q \longrightarrow A & : & N^Q_{\text{dotp}} \\
(\mathrm{A}_9) & A \longrightarrow S & : & N^Q_{\text{sessid}}, N^Q_{\text{dotp}} \\
(\mathrm{A}_{10}) & S \longrightarrow Q & : & N^Q_{\text{sessid}}, N^Q_{\text{dotp}} \\
(\mathrm{A}_{11}) & Q \longrightarrow S & : & N^Q_{\text{sessid}}
\end{array}$$

The protocol can be explained as follows: The client initially requests an authentication session with the service provider S. This request is forwarded to the key server, including the name of the client and her phone number. The key server Q then sends a session identifier N^Q_{sessid} to both S and A in the message (A_3) and (A_4). The client sends a signal to the key server Q to authenticate herself and includes N^Q_{sessid}. The key server Q then sends two transaction keys, the current (old) key K_{AQ} and a fresh key K'_{AQ} to the client A, together with a nonce N^Q_{chall}. Prior to (A_7), the client decrypts the client reference that was encrypted at the end of the registration (C_1) or at the end of the previous authentication as we shall see later in (C_{12}). Then the security code is reconstructed by the client: $\mathsf{H}[T_{\text{pin}}, T_{\text{imei}}, N^A_{\text{cref}}, S]$, and the challenge is encrypted with the security code and sent to Q. The key server Q replies by sending display One Time Password (OTP) N^Q_{dotp} to A, that forwards it together with the session identifier to S. At the end both N^Q_{sessid} and N^Q_{dotp} are sent to the key server Q, and then N^Q_{sessid} is sent back to S. The two final messages seem superfluous, but they are included in order to enable the service provider to pick up the authentication result. The key server must be certain that the session identifier and display OTP have not been changed during a session.

Analogous to registration, the authentication protocol ends with a sequence of local actions. Since the client deleted its local occurrence of the security code in (C_5), it had to rebuild it prior to message (A_7). Authentication is therfore succeded by:

$$\begin{array}{ll}
(\mathrm{C}_6) & A \text{ performs } E(K'_{AQ} \ : \ N^A_{\text{cref}}) \\
(\mathrm{C}_7) & A \text{ removes } E(K_{AQ} \ : \ N^A_{\text{cref}}) \\
(\mathrm{C}_8) & A \text{ removes } K_{AQ} \\
(\mathrm{C}_9) & A \text{ removes } K'_{AQ} \\
(\mathrm{C}_{10}) & A \text{ removes } T_{\text{pin}} \\
(\mathrm{C}_{11}) & A \text{ removes } N^A_{\text{cref}} \\
(\mathrm{C}_{12}) & A \text{ removes } \mathsf{H}[T_{\text{pin}}, T_{\text{imei}}, N^A_{\text{cref}}, S]
\end{array}$$

The client reference is protected by the fresh transaction key K'_{AQ} in (C_6), the old occurrence is removed (C_7), including the elements in the security code and the transaction keys ($\mathrm{C}_8 - \mathrm{C}_{12}$).

3 Analysis process

The analysis of the security critical aspects of the application was divided into several stages. In this section we give an overview of the analysis process, and an example of a phishing attack on the application. A large collection of potential and real attacks can be found in [7]. First, we specified the entire application in the formal protocol language. This was definitely the most exhausting part of the work, since the documentation was incomplete, misleading and sometimes inconsistent. The documentation included models and figures showing the message flows in the protocols, and informal explanation of the technology used.

3.1 Security goals

The security goals of the Key Server application was not specified explicitly in the documentation when the security analysis started. Both the design and the implementation of the prototype was performed by a handful of experienced software engineers, and the overall requirements of authenticity and confidentiality were implicit shared knowledge of the development team. For the analysis of the security it turned out to be important to be fully explicit about the requirements of the application. Security goals might be either high or low level requirements. Low level goals, like for instance the confidentiality or authenticity of certain nonces or keys, can be considered as low level cryptographic mechanisms to secure that high level goals are achieved. The high level security goals of the application is first to authenticate a user to a key server and then to authenticate the user to service providers. In Table 1, the low level security goals are formulated and categorized. The goals were stated explicitly by the application designers. A data unit e fulfills a *confidentiality* goal if e is not disclosed to an illegitimate agent during a protocol run. A data unit e fulfills an *authenticity* goal if e is not compromised during a protocol run, that is the sender or receiver can be ascertained that e is not changed in any way.

	Security Goals	
	Authenticity	Confidentiality
Regis-tration	$\mathsf{H}[T_{\text{pin}},T_{\text{imei}},N^A_{\text{cref}},S]$	N^A_{cref}, N^S_{acode} $\mathsf{H}[T_{\text{pin}},T_{\text{imei}}, N^A_{\text{cref}}, S]$
Authen-tication	N^Q_{chall}	K_{AQ}, K'_{AQ} N^Q_{chall}, N^A_{cref}

Table 1 Detailed security goals proposed by the company.

3.2 Simulations

After a coherent and consistent specification was obtained, the protocols were simulated. Simulations were performed in three stages: First an environment containing only good agents was configured and simulated. Then the simulator was configured with a *passive attacker* (also called eavesdropper or man-in-the-middle) that intercepted all messages on the network. The passive attacker is a man-in-the-middle, that intercepts and forwards every message. In addition the attacker extracts information from the messages that it forwards. Hence the passive attacker can only break confidentiality goals of a given protocol if the content is not protected by encryption or if the appropriate keys are disclosed. Finally, several *active attacks* were specified and simulated, these attacks were, to some extent, based on the passive attacks.

Since attacks can be specified analogous to protocols and run inside agents, active attacks might be launched as well: We launched two kinds of active attacks: *application layer attack* and *corrupt SSL/TLS attacks*. In the former case we assumed that the SSL/TLS channel was broken, and every message was sent in plain text. Each of the low level security goals shown in Table 1 were broken in a large collection of attacks (83 active attacks). These active attacks included *fake client*, *fake service provider*, and *fake key server*. In the latter case the application layer code is wrapped inside a SSL/TLS session, but the SSL/TLS session is corrupt: the client establishes a connection to an attacker, although the application layer content is intended to the good agent. Then the attacker establishes a new SSL/TLS connection with the intended server forwarding the same message content or some modification of the content. The TLS channel has been modeled as an encryption with the TLS session key, hence an application layer message

$$A \longrightarrow B \ : \ M,$$

was embedded in TLS by encryption of the message content M, using the specific TLS session key

$$A \longrightarrow B : E(K^{\text{TLS}}_{AB} \ : \ M).$$

3.3 Concrete flaws

The initial investigation revealed problems or errors concerning five data expressions in the two principal protocols, two of these pieces of data were cryptography units and the remaining three where agent names. The initial specification did not explicitly state that the security code was transferred in message R_5. It was also a bit unclear whether and how the client reference should be encrypted. The missing variables for agent names included the key server variable Q in message R_1, the client identity A in message R_3, and finally the service provider identity S in the security code throughout the specification. The missing key server name Q was particularly interesting, since the inclusion of the key server variable entailed that the application could be extended to include a collection of key servers instead of one single server. In addition, validation and simulation revealed that four essential messages were lacking entirely (A_1, A_9, A_{10}, and A_{11}) and one was incorrectly specified, the sender and receiver were both incorrect in message A_8.

3.4 Phishing attacks

Phishing attacks are commonly regarded as "social" attacks. Phishing attacks occur since users are mislead to expose confidential information to an agent that is not the intended receiver. A phishing attack can be described by replacing a normal transmission:

$$A \longrightarrow B : E(K^{\text{TLS}}_{AB} \ : \ F)$$

with the interception and forwarding:

$$A \longrightarrow I(B) : E(K^{\text{TLS}}_{AI} \ : \ F)$$
$$I(A) \longrightarrow B : E(K^{\text{TLS}}_{IB} \ : \ F')$$

where F' might be a modification of the original content F. Following this modification, it is possible to construct a phishing attack on both the registration and authentication protocol.

In the case of the registration protocol, the client might be fooled to use a wrong TLS key denoted K^{TLS}_{AI}. The attacker then establishes a connection K^{TLS}_{IS} to the service provider. We assume that the client is unable to recognize this fact: although the wrong key is used, the use of the wrong key is not visible or understandable to the client. The attacker receives each of the messages from the client as plain-text. We also assume that the key server Q is fooled in initiating a TLS session with the intruder instead of the K^{TLS}_{QI}. Then the attacker I establishes a TLS-connection with the agent A, where I impersonates the key server Q. Since the TLS authentication is one-way, the client A can not discover that the channel is corrupt. The attack can be specified as given in Figure 3. From this attack we see that every security critical unit is compromised, except the client reference, the PIN code and the IMEI.

$$
\begin{array}{lll}
(R^{\mathrm{P}}.1.a)\ A \longrightarrow I(S) & : & E(K_{AI}^{\mathrm{TLS}} : T_{\mathrm{service}}, T_{\mathrm{phone}}, Q) \\
(R^{\mathrm{P}}.1.b)\ I(A) \longrightarrow S & : & E(K_{IS}^{\mathrm{TLS}} : T_{\mathrm{service}}, T_{\mathrm{phone}}, Q) \\
(R^{\mathrm{P}}.2.a)\ S \longrightarrow I(A) & : & E(K_{IS}^{\mathrm{TLS}} : N_{\mathrm{acode}}^{S}) \\
(R^{\mathrm{P}}.2.b)\ I(S) \longrightarrow A & : & E(K_{AI}^{\mathrm{TLS}} : N_{\mathrm{acode}}^{S}) \\
(R^{\mathrm{P}}.3.a)\ S \longrightarrow I(Q) & : & E(K_{SQ}^{\mathrm{TLS}} : A) \\
(R^{\mathrm{P}}.3.b)\ I(S) \longrightarrow Q & : & E(K_{SQ}^{\mathrm{TLS}} : A) \\
(R^{\mathrm{P}}.4.a)\ Q \longrightarrow I(A) & : & E(K_{QI}^{\mathrm{TLS}} : K_{AQ}) \\
(R^{\mathrm{P}}.4.b)\ I(Q) \longrightarrow A & : & E(K_{IA}^{\mathrm{TLS}} : K_{AQ}) \\
(R^{\mathrm{P}}.5.a)\ A \longrightarrow I(Q) & : & E(K_{IA}^{\mathrm{TLS}}, \mathsf{H}[T_{\mathrm{pin}}, T_{\mathrm{imei}}, N_{\mathrm{cref}}^{A}, S], \\
 & & E(\mathsf{H}[T_{\mathrm{pin}}, T_{\mathrm{imei}}, N_{\mathrm{cref}}^{A}, S] : N_{\mathrm{acode}}^{S})) \\
(R^{\mathrm{P}}.5.b)\ I(A) \longrightarrow Q & : & E(K_{QI}^{\mathrm{TLS}}, \mathsf{H}[T_{\mathrm{pin}}, T_{\mathrm{imei}}, N_{\mathrm{cref}}^{A}, S], \\
 & & E(\mathsf{H}[T_{\mathrm{pin}}, T_{\mathrm{imei}}, N_{\mathrm{cref}}^{A}, S] : N_{\mathrm{acode}}^{S}))
\end{array}
$$

Fig. 3 Phishing the registration protocol.

$$
\begin{array}{llll}
(A^{\mathrm{P}}.2.1) & I(A) \longrightarrow S & : & E(K_{IS}^{\mathrm{TLS}''} : T_{\mathrm{auth}}) \\
(A^{\mathrm{P}}.2.2.a) & S \longrightarrow I(Q) & : & E(K_{SI}^{\mathrm{TLS}'''} : A, S, T_{\mathrm{phone}}) \\
(A^{\mathrm{P}}.2.2.b) & I(S) \longrightarrow Q & : & E(K_{IQ}^{\mathrm{TLS}'''} : A, S, T_{\mathrm{phone}}) \\
(A^{\mathrm{P}}.2.3.a) & Q \longrightarrow I(S) & : & E(K_{IQ}^{\mathrm{TLS}'''} : N_{\mathrm{sessid}}^{Q}) \\
(A^{\mathrm{P}}.2.3.b) & I(Q) \longrightarrow S & : & E(K_{SI}^{\mathrm{TLS}'''} : N_{\mathrm{sessid}}^{Q}) \\
(A^{\mathrm{P}}.2.4) & Q \longrightarrow I(A) & : & N_{\mathrm{sessid}}^{Q} \\
(A^{\mathrm{P}}.2.5) & I(A) \longrightarrow Q & : & E(K_{IQ}^{\mathrm{TLS}'} : T_{\mathrm{auth}}, N_{\mathrm{sessid}}^{Q}) \\
(A^{\mathrm{P}}.2.6) & Q \longrightarrow I(A) & : & E(K_{IQ}^{\mathrm{TLS}'} : N_{\mathrm{chall}}^{Q}, S, K_{AQ}, K'_{AQ}) \\
(A^{\mathrm{P}}.2.7) & I(A) \longrightarrow Q & : & \\
 & \multicolumn{3}{l}{\quad E(K_{IQ}^{\mathrm{TLS}'} : E(\mathsf{H}[T_{\mathrm{pin}}^{A}, T_{\mathrm{imei}}^{A}, N_{\mathrm{cref}}^{A}, S] : N_{\mathrm{chall}}^{Q}))} \\
(A^{\mathrm{P}}.2.8) & Q \longrightarrow I(A) & : & E(K_{IQ}^{\mathrm{TLS}'} : N_{\mathrm{dotp}}^{Q}) \\
(A^{\mathrm{P}}.2.9) & I(A) \longrightarrow S & : & E(K_{IS}^{\mathrm{TLS}''} : N_{\mathrm{sessid}}^{Q}, N_{\mathrm{dotp}}^{Q}) \\
(A^{\mathrm{P}}.2.10.a) & S \longrightarrow I(Q) & : & E(K_{SI}^{\mathrm{TLS}'''} : N_{\mathrm{sessid}}^{Q}, N_{\mathrm{dotp}}^{Q}) \\
(A^{\mathrm{P}}.2.10.b) & I(S) \longrightarrow Q & : & E(K_{IQ}^{\mathrm{TLS}'''} : N_{\mathrm{sessid}}^{Q}, N_{\mathrm{dotp}}^{Q}) \\
(A^{\mathrm{P}}.2.11.a) & Q \longrightarrow I(S) & : & E(K_{IQ}^{\mathrm{TLS}'''} : N_{\mathrm{sessid}}^{Q}) \\
(A^{\mathrm{P}}.2.11.b) & I(Q) \longrightarrow S & : & E(K_{SI}^{\mathrm{TLS}'''} : N_{\mathrm{sessid}}^{Q})
\end{array}
$$

Fig. 4 Phishing the authentication protocol (fake client).

In Figure 4 the attack on the registration is extended to an attack on the authentication protocol. The intruder is impersonating the client and fooling both the key server and the service provider to believe that an appropriate authentication session has been performed.

4 Conclusion

The investigation revealed several weaknesses of the original application, some of these were known in advance and some were discovered through the analysis. The analysis showed that the documentation was incomplete, cryptographic entities were lacking, messages were lacking, and the use of SSL/TLS sessions were not explicitly specified. Even more important, it turned out that many security goals were not specified explicitly. Some unexpected architectural decisions that had not been investigated previously, were discovered through simulations. The resistance against phishing attacks could be discussed in a very precise manner due to simulations based on formal specifications of the application.

The collaboration with the company was a major success factor for the investigation: In general the company was positive to a formal analysis, the development team were skilled computer scientists that could easily understand the approach and goal of the security analysis, and the key personnel were always available answering questions and commenting on the formal specifications written during the project.

Both validation and simulation turned out to be beneficial, both for the checking whether the documentation was correct and for the security analysis of the protocol. By making the protocols and potential attacks transparent, the application has been improved considerably by semi public benchmarking and tool analysis. A large number of potential attacks has been constructed, and the application has been simulated to see the effects of these attacks. Hence our work stand in contrast to the current attitude often found in Bank business, that favour 'security by obscurity': for several Internet banks, secrecy of the technical design is assumed to be important to the security of the application. Yet, trust in applications should be based on open evaluation, not by hiding technical solutions that are error-prone or poorly understood. Even more important: keeping documentation about an application confidential does not save it from attacks!

Acknowledgements The authors would like to thank Petter Taugbøl, Wolfgang Leister, Knut Eilif Husa, Anders Bolander and Erik Lindmo for comments and discussion on earlier versions of this paper.

References

1. Markus Jakobsson, Steven Myers: Phishing and Countermeasures, 2007, John Wiley & Sons
2. Enhancing One-Time Passwords for Protection against Real-Time Phishing Attacks, (2006), obtained from RSA at `www.rsasecurity.com/`
3. Protecting Against Phishing by Implementing Strong Two-Factor Authentication (2006)
4. Michael Burrows and Martín Abadi and Roger Needham, A Logic of Authentication, Technical Report 39, Digital Systems Research Center, (1989)
5. Kjell J. Hole and Vebjørn Moen and Thomas Tjøstheim, Case Study: Online Banking Security, IEEE Security and Privacy, Vol. 4, No. 2, pp. 14–20, (2006), IEEE Educational Activities Department
6. Anders Moen Hagalisletto, Attacks are Protocols Too, Proceedings of The Second International Conference on Availability, Reliability and Security , pp. 1197 – 1206
7. Anders Moen Hagalisletto, Analyzing two-factor authentication devices, University of Oslo, Report 357, ISBN 82-7368-314-1, June (2007).
8. John C. Mitchell, Vitaly Shmatikov and Ulrich Stern, Finite-State Analysis of SSL 3.0 and Related Protocols, Proceedings of the DIMACS Workshop on Design and Formal Verification of Security Protocols, september 1997,
9. Lawrence C. Paulson, Inductive Analysis of the Internet Protocol TLS, ACM Transactions on Information and System Security, august, Vol 2, No. 3, pp. 332-351, (1999)
10. Kazuhiro Ogata and Kokichi Futatsugi, Equational Approach to Formal Analysis of TLS, ICDCS '05: Proceedings of the 25th IEEE International Conference on Distributed Computing Systems (ICDCS'05), pp. 795–804, (2005)