Knowledge-Based Systems 59 (2014) 149–158

Contents lists available at ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys

On securing online registration protocols: Formal verification of a new proposal

Jesus Diaz *, David Arroyo, Francisco B. Rodriguez

Grupo de Neurocomputación Biológica, Departamento de Ingeniería Informática, Escuela Politécnica Superior, Universidad Autónoma de Madrid, Spain

ARTICLE INFO

Article history:
Received 27 May 2013
Received in revised form 2 December 2013
Accepted 14 January 2014
Available online 5 February 2014

Keywords:
Protocol security
Automated verification
EBIA
Security-by-design
Digital identity

ABSTRACT

The deployment of Internet based applications calls for adequate users management procedures, being online registration a critical element. In this respect, Email Based Identification and Authentication (EBIA) is an outstanding technique due to its usability. However, it does not handle properly some major issues which make it unsuitable for systems where security is of concern. In this work we modify EBIA to propose a protocol for users registration. Moreover, we assess the security properties of the protocol using the automatic protocol verifier ProVerif. Finally, we show that the modifications applied to EBIA are necessary to ensure security since, if they are removed, attacks on the protocol are enabled. Our proposal keeps the high usability features of EBIA, while reaching a reasonable security level for many applications. Additionally, it only requires minor modifications to current Internet infrastructures.

1. Introduction

Creating usable systems is certainly a subject of critical importance for Internet applications aiming to reach a high acceptance among its users. However, combining usability with security is normally a source of difficulties. In the area of online registration, which comprises a critical component of Internet based systems, Email Based Identification and Authentication (EBIA) is almost certainly one of the techniques that stands out among the set of alternatives [1]. EBIA uses email addresses as identifiers, and as authenticators the fact of accessing to URLs contained within email messages sent to those addresses. Its main advantages are usability and ease of deployment, while its major drawback is security [2].

In the registration process of EBIA-based systems (shown in Fig. 1) it is assumed that only the owner of the email account can access the activation link sent within registration email messages. However, this has a well-known security problem [2], for emails are typically sent over an insecure channel (with no encryption whatsoever). In this matter it should be noticed that the SSL/TLS connection between user and email server does not suffice to avoid eavesdropping unless emails are encrypted, since the rest of the path followed by the email is not always encrypted. As a result, a passive attacker might wait until her victim initiates registration, and then access the activation link by eavesdropping the email, possibly setting a new password or something similar (this sequence is shown following the initial step 1a in Fig. 2). Alternatively, an active attacker might try to register herself using an email account that she does not own, since she does not need access to the email account in order to read the email message contents, as shown in the path initiated with step 1b in Fig. 2. Therefore, the actual flaw of EBIA is to assume that accessing the link sent within the registration emails is equivalent to a proof of ownership of the associated email account.

Our proposal is to convert EBIA's authenticating action (i.e., accessing the registration link) into an explicit acknowledgment message sent by a trusted entity. In fact, the protocol structure provides an ideal candidate for this role, namely, the email provider. More specifically, the Mail Servers that are accessed securely (it is a common practice nowadays to protect accesses to Mail Servers with SSL/TLS) by the email account owners to fetch the email messages in their in-boxes. By providing the Mail Servers with digital identities trusted by the registration server, those servers can send authenticated acknowledgment messages after receiving an instruction from the user.

The rest of the paper is organized as follows. In Section 2 we analyze several alternatives/extensions to EBIA toward solving the above explained security drawbacks. Based on the insight gained from this analysis, we present an EBIA-based registration system in Section 3. The security of this scheme is evaluated in Section 4 by using the automated protocol verifier ProVerif [3]. In Section 5 the security assessment is complemented by discussing the minimality of the protocol, i.e., by underlining the needs of each component of the protocol and pinpointing the subsequent attacks

* Corresponding author.
E-mail address: j.diaz@uam.es (J. Diaz).

http://dx.doi.org/10.1016/j.knosys.2014.01.011
0950-7051/

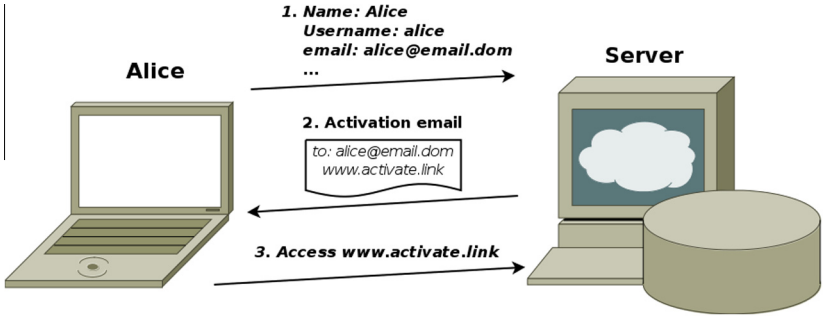

Fig. 1. Behavior of EBIA systems. (1) The user sends the request with her data. (2) The server validates the data and sends back an activation link via email. (3) The user accesses the activation link.

enabled in case of either omitting some of them or changing the interrelationship order. We continue in Section 6 with an analysis of the additional costs, trust relationships, modifications over the existing infrastructures and usability. Finally, the article ends with the conclusion in Section 7.

2. A brief security analysis of EBIA-based registration protocols

Several alternatives have been proposed to solve or reduce the impact of EBIA's security limitations, typically by incorporating some kind of multifactor and/or multichannel method [4]. In [5], an authentication protocol is proposed intended to provide single use authenticators through a multichannel technique, with the aim of reducing the effect of possible attacks. The security of the scheme is claimed to improve the performance of EBIA by adding a token sent via SSL ("SSL token") in the same session established by the registering user during the authentication request. Besides sending this token via SSL, an extra token is sent unencrypted via email. The related digital identity is only generated if the user sends back to the server both tokens. The intuition behind this approach is that, by using SSL, only the user who started the authentication request receives the "SSL token". Thus, if this user also proves knowledge of the token sent via email, it is assumed that the user who started the session is the owner of the email account. However, even with SSL in use, this approach is vulnerable to the attack on EBIA illustrated in Fig. 2 (attack initiated with step 1b). Although it is an attack that assumes that the attacker takes an active role (i.e., it is not limited to just observing the communications), its security is not higher than that of classic EBIA. In detail, an active attacker starting the registration process would just have to eavesdrop the activation email to obtain a legitimate identity. For the sake of clarity, we have implemented a ProVerif model of this protocol to show the mentioned attack by executing the code available from [6]. This shortcoming is addressed in [5] by combining several email accounts. In this new setup, the challenge for an attacker is more complicated, since she has to break into several Mail Servers. Nevertheless, this also erodes the scheme's usability as users have to manage more than one email address, and thus they can be reluctant to adopt this alternative. As a result, a third possible implementation is proposed in [5] consisting first of the encryption of the email using a key sent jointly with the SSL token, and secondly by routing the resulting message via an anonymizing network. Therefore, even though the attacker can start the SSL session and get the SSL token plus the key used to encrypt the email, allegedly she cannot gain possession of the email because it is sent via an anonymizing network. However, if the

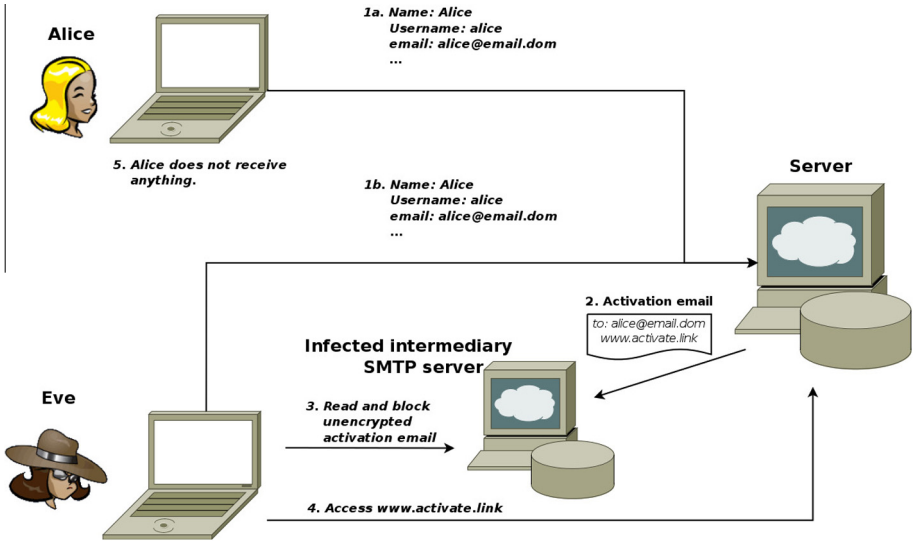

Fig. 2. Attacks to EBIA systems. The attack initiated with step 1a depicts a possible attack by passive adversaries. The attack initiated with step 1b depicts a possible attack by active adversaries. Steps 2–5 are the same for both attacks. (1a) Alice requests registration to the Web Server. (1b) Eve starts registration on behalf of Alice. (2) The server sends the activation email to Alice, containing the activation link, which passes through an infected intermediary server under Eve's control. (3) Eve intercepts and blocks the activation email. (4) Eve activates the account in Alice's behalf. (5) Alice will probably just think that an error occurred.

communications with the exit (entry) point from (to) the anonymizing network are not protected, this option is still insecure.

In [7] EBIA takes an important role in a method to avoid phishing attacks. The method is a two-factor authentication procedure based on browser bookmarks and fragmented identifiers. However, the setup of this bookmark relies directly on a verification code sent to the registering email address. To secure this step, the author states that this email should be "secure in authentication and in content" (hence, signed and encrypted). Otherwise, it would also be vulnerable to the same attacks as EBIA. Indeed, encryption and sender authentication in the activation email is a valid option (in fact, it would probably be the best one). Nevertheless, we prefer to avoid it, mainly because of two reasons: first, because users seem reluctant to use email encryption [8–10]; and second, if the emails can be sent in such a way then the authenticating token can be sent straight away within the email. Certainly, only the legitimate owner of the email account is going to be able to read it (this is guaranteed by encryption), and thus any extra step to provide her with a valid token seems unnecessary.

Other widely known system using two-factor authentication is Google. For setting up a new account, Google requires to introduce a mobile phone number in order to send a code via SMS (or a voice call). In this fashion, user verification is fulfilled by providing the received code. This protocol actually increases the overall system security at the cost of some loss in usability, but it is not a *perfect* solution either since the communication channels do not offer point-to-point security [11]. Therefore, like in the case of using emails it is not correct to assume that the communication channels are perfectly secure.

3. The proposed protocol

We proceed to explain the proposed protocol. First, we introduce its entities and communication channels. Subsequently, we explain the main modifications introduced over the classic EBIA, and finish with a description of the sequence of message exchanges that occur within a normal protocol run. As for the entities, three of them take part in our protocol:

User: Represents any user who starts the registration process. Nothing is assumed about the users besides that they own an email account with some email provider.
MS: A Mail Server of the domain where the user owns an email account. The Mail Server is assumed to own a digital identity trusted by the Web Server.
WS: The Web Server providing the registration service.

Within a protocol run, these three entities mostly communicate between themselves via server authenticated SSL/TLS sessions. Additionally, the WS will use once an unreliable, unencrypted and unauthenticated channel to send the email messages, and the MS will also use once an authenticated channel.

3.1. Modifications to classic EBIA

Our protocol closely resembles the original EBIA. However, it adds two messages that explicitly provide the guarantees implicitly assumed by EBIA and the related systems studied above. In fact, EBIA works by assuming that the legitimate owner of the email address receives the activation link. This is similar to taking for granted that the sender actually knows that the email has reached the intended recipient, and that accessing the activation link acknowledges having received that email. Our approach is to convert this implicit assumption into an explicit message, hereafter noted as `ACK`, which provides *explicit reliability* in the sense that the WS has the certainty that the legitimate user, and no other entity, has confirmed having received the registration email. There is only one trust point, specifically, the server providing this registration service must trust the Mail Server. This implies that when the MS sends to the WS a message acknowledging that the owner of a certain email account within its domain has received a certain email, the WS trusts it. It is not a too strong assumption, provided that the communication between the user and the MS is secured (which is the most common practice today [12]). Again, note the difference between this assumption and the one made by EBIA and its derivatives. In the latter, the WS assumes that, if a link that has been sent via email to a given email address is accessed, then the intended recipient (and no one else) has actually received it; in the former, it assumes that if a (trusted) mail provider reports that one of its users has received an email, it must be so.

In order to guarantee that only the same user who started the registration process will be able to complete it, we add an extra token, which we call the SSL registration ticket. It is sent via the first SSL/TLS session established when the User first requests to be registered. After having verified the email address, if someone provides that same SSL ticket, the WS will be certain that (1) she is the same user who started the registration; and (2) she is the legitimate owner of the introduced email address.

3.2. Protocol description

First, the User requests registration by establishing a server authenticated SSL/TLS session with the WS and sending her email address. Then, the WS uses the same SSL/TLS session to send back the *SSL registration ticket* (which must be unique to this request) and a nonce, the *email registration nonce* (henceforth, *email nonce*). Furthermore, the WS sends the same *email nonce* to the user through the MS as an unencrypted email message. Subsequently, the user securely connects with her MS and waits for the email. When it arrives, she verifies that the nonce received via email matches the one received via SSL/TLS. If so, the user sends the MS a request to acknowledge the receipt of the message. Consequently, the MS digitally signs the ACK, and sends it to the WS. At this point, the WS has the certainty (because it trusts the MS) that the legitimate owner of the email address has received and acknowledged the email with that concrete nonce. Finally, when the user sends back the SSL/TLS registration ticket, the WS issues the new user's identity. The shape that this identity takes may vary depending on the needs of each system and its intended functionality. It might be a PKCS12 token containing an X.509 certificate and its associated private key or it might just be a new password for a new website. Even more advanced and complex protocols for identity negotiation could be executed securely within this last SSL/TLS session, with the guarantee that the user is the owner of the specified email address. A general view of the protocol is shown in Fig. 3. In the next section we proceed with an in-depth security analysis of the proposed protocol.

4. Formal security analysis

The security properties that a typical registration protocol needs to fulfill are confidentiality of the conveyed identity, and authenticity in the sense that the communicating parties prove who they are: the server must authenticate itself to avoid phishing; and the user must prove that she is who she claims to be. In order to prove these security properties, we rely on ProVerif,[1] an automatic protocol verifier. Certainly, formal methods applied to the security verification of communication protocols have been

[1] http://www.proverif.ens.fr/.

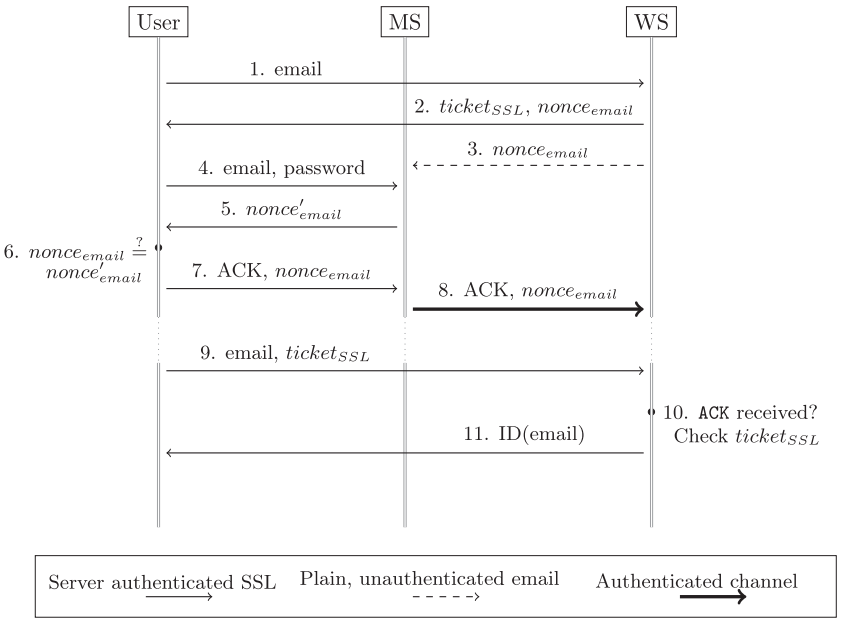

Fig. 3. Sequence diagram of a normal run of the protocol. User stands for the user requesting registration, MS stands for Mail Server and WS stands for the Web Server providing the registration service. The thin continuous lines represent SSL sessions with server authentication; the dashed line represents an unencrypted and unauthenticated channel (classical email); the thick continuous line represents an authenticated channel. (1) The user sends her email to request registration. (2) The Web Server sends back the SSL ticket, jointly with an email nonce. (3) The Web Server sends an email to the specified email address, containing the email nonce. (4) The user logs in her email account, being thus authenticated with the Mail Server. (5) The Mail Server sends the user the received email. (6) The user checks that the email nonce received in step 2 and the email nonce contained in the email are the same. (7) If positive, the user instructs the Mail Server to send an ACK to the received email. (8) The Mail Server sends a signed ACK. (9) The user sends back the SSL ticket obtained in step 2. (10) The Web Server verifies that the associated email has been acknowledged. (11) Finally, the Web Server issues the identity.

proved to be a very helpful technique [13–15], since they allow and ease the detection of important (and many times convoluted) security flaws in their design. ProVerif accepts a model, in a variant of the applied pi calculus, of the protocol under perusal, and converts it to Horn clauses [16] on which automated reasoning techniques are applied in order to prove if the required properties hold (actually, ProVerify accepts several input formats: typed pi calculus, pi calculus, typed Horn clauses, and Horn clauses, although only the first one is currently being actively maintained [3]). For further details on the tool and the theory behind ProVerif, we refer to [17,18,3,19,20]. For our work, suffice it to say that ProVerif provides *soundness*, i.e., when it says that certain property is satisfied in the received model, then that model really guarantees that property. Finally it is also important to emphasize that, during our analysis, we adopt the Dolev–Yao model, which assumes perfect cryptographic primitives [21]. This is the usual practice in this kind of analysis, since we are worried about the protocol logic, and not its implementation [14].

For the verification of our protocol, we have applied a preliminary version of the methodology introduced in [22]. However, in this work instead of showing in detail the application of the methodology over the proposed protocol, we center our attention in the formalization of the protocol. Thus, we show that the obtained model is a valid abstraction of the protocol (step 7 in [22]), which endorses the subsequent formal verification of the main objective of the protocol (step 8 in [22]), i.e., the distribution of digital identities. Additionally, we provide a much deeper analysis by showing that all the components of the protocol are necessary for providing the required security properties. Moreover, this detailed analysis allows us to compare our proposal with the existing alternatives, evidencing that it solves their security problems at a minimal cost. In the following subsections we explain how we have modeled the protocol in terms of the participating entities and communication channels. For the sake of readability, instead of explaining the three entities, we center our attention on the User and MS processes and just make a few comments on the WS logic. Certainly, given the details of User and MS, it is easy to grasp the behavior of the WS. Additionally, the source code of the complete model, ready to be processed with ProVerif, is publicly available in [6].

4.1. Communication channels

Our model makes use of three types of communication channels. They are not (or do not need to be) different communication channels from the perspective of their physical properties. Instead, we use the term "communication channel type" to refer to channels with different security properties:

SSL/TLS. Channels by means of which users establish server authenticated sessions with both servers. To model the key negotiation, we use a process that creates a fresh symmetric key after receiving a request from the user, sent via a private channel (i.e., only accessible to legitimate users and servers). The generated symmetric key is sent back to the user via the previous private channel. In this case, the usage of ProVerif's private channels allows the two parties to establish a new key secretly, just like a real SSL/TLS key negotiation. However, in order to make this abstraction a faithful model of our scenario, we also have to allow the attacker to establish SSL/TLS sessions with the servers. For that purpose, we create an additional process that instead of receiving "SSL negotiation" requests from a private channel, receives them from the public channel, accessible to the attacker. Note that this approach enables both legitimate users and attackers to actually establish secure sessions. Also, since the other end of the communication in this exchange is always one of the two servers (at the user/attacker's choice), this successfully emulates an SSL key negotiation with server authentication.

Conventional email. When the WS receives a registration request, it sends a conventional email to the specified email address. To emulate the security properties of emails, this message is sent via the public network, in plaintext and without any authentication. This allows the attacker to access its contents, modify them and even generate fake emails.

Authenticated channel. The MS, upon receiving a request sent by the User to send the `ACK`, creates it, digitally signs it, and sends it back to the WS over the public network, without encryption. Hence, this approach satisfactorily emulates an authenticated channel.

4.2. The User process

This process represents the user willing to be registered in a new site. In our model the user has three attributes: a hostname, an email account, and a password for that email account. They are represented in the code in Listings 1–5 as `u`, `e` and `pwd`, respectively. The hostname `u` represents the "name" or IP address of the machine of the user. It is not a field strictly necessary for our protocol, but it helps to determine the messages source. Note that it is declared as a new name and always sent encrypted, but we make it public by sending it over the public channel `net` in order to model that the attacker might know the hostname (or IP address) of its victims. The email `e` is the actual identifying information used by our protocol, and it is also used to simulate the fact that each specific WS may restrict the admissible email domains. Thus, legitimate users (i.e., users with valid emails) declare their email as a new name within their code, and insert it into a table named `emails` used by the WS to check that it is a valid email address. The email is also inserted along with the associated password `pwd` in a table named `msaccounts` used by the MS to authenticate its email users. Since tables are not accessible to the attacker in ProVerif, this successfully models our assumption that only the legitimate owner of an email account can prove ownership to the MS. Additionally, the email address is immediately sent out via the public network to emulate the very possible fact that an attacker may know the email address of its victim. The code associated to all these initializations is shown in Listing 1.

Once all the user's internal data have been set, the user can start an SSL/TLS session by sending out a message through the `privateuserchannel` (in line 3 of Listing 2). The first field of this message specifies the host initiating the session, the second field determines the other end of the communication (`ws` comes from Web Server), and the third field is a *nonce* to avoid mixing up keys generated for different SSL sessions (this is possible due to a subtlety in ProVerif, that allows the reasoning engine to resend messages sent via private channels). Once the SSL session has been established, the user sends to the WS her email address via SSL in the message in line 8 of Listing 2. Here, the second and third fields determine the sender and receiver, the last is the email, and the first is just a tag to ease the debug of the protocol (the number indicates the order in which these messages are sent). In the last step of this SSL session (lines 12–14 of Listing 2) the user receives the ticket generated by the WS, along with the nonce that will allow the user to verify the email that the WS has already sent at this point. Only messages with the expected format, and encrypted using the SSL key negotiated before will be accepted at this point. Since the key has been negotiated in with server authentication, the user has guarantees that it indeed comes from the WS.

Once the User receives the SSL ticket and the email nonce, she waits for an email from the WS. Thus, she establishes an SSL connection with her MS and gains access through authenticating herself by means of email address and the associated password (line 9 in Listing 3). If the MS has received the email, it sends it back to the user, also encrypted via the same SSL session. The user receives and decrypts it (lines 10 and 14, respectively) and in case the *nonce* contained within the email is the same than the *nonce* received via SSL (this is specified with the `=ne` syntax in line 14), then she sends to the MS an instruction to send an authenticated `ACK` message back to the WS. This last step is performed in line 17 of Listing 3.

In the last step the user has to send back the SSL token to the WS in order to complete the registration. To do so, the user establishes a new SSL session in lines 3 and 4 of Listing 4. Indeed, this models the probable fact that the user could need to establish a new SSL session with the WS instead of using the previous one (e.g., the user can perform this last step the next day after initiating the registration, or after rebooting her machine). Finally, the user sends the SSL ticket back to the WS (in line 6). When the WS receives the ticket, it will check that it matches the ticket sent to this user and that the `ACK` sent by the MS has already been received. This last check is of the utmost importance in order to be sure that the user is the owner of the email address. Subsequently, the identity is generated and sent to the user, who receives and decrypts it in lines 7 and 9 of Listing 4. Again, given that the SSL session has been established with server authentication, the user can correctly assume that the new identity actually comes from the WS.

Therefore, Listings 1, 2, 3 and 4 compose a realistic model of the typical users that would take part in our registration protocol.

4.3. The Mail Server

The MS process emulates the behavior of the Mail Servers that take part in our protocol. The initialization of this process just consists of receiving two parameters: the MS private key, and the WS public key. Both keys are generated in the "main" process that initializes the environment and launches the processes corresponding to each entity. Indeed, this is a reasonable way of modeling the *real world* scenario in which each trusted server has its own private key, certified by some Certification Authority, and the public key is accessible to anyone else. The MS is "invoked" by receiving a message containing a nonce, and having as final destination a certain email address. This is performed in line 5 of Listing 5. Note that it is sent via the public channel (`net`) without neither encryption nor digital signatures, which models the real scenario where the MS cannot determine the email source (it could be either the WS or an attacker) and eavesdropping is possible (since an attacker can read the email sent by the WS).

After receiving the message with the nonce, the MS waits until a new SSL session is established and someone provides a valid email account along with the associated password (checked by accessing the `msaccounts` table). This is done in lines 11–15 in Listing 5. When that occurs, the MS sends to the user (line 19 in Listing 5),

```
1  let userprocess =
2
3          new u:Host; out(net,u); (* The machine name *)
4          new e:Email; insert emails(e); out(net,e);
5          new pwd:Password; insert msaccounts(e,pwd);
```

Listing 1. User process code: initialization.

```
1   (* Establish the first SSL session *)
2   new n1:Nonce;
3   out(privateSSLuserchannel, (u,ws,n1));
4   in(privateSSLuserchannel, (=ws,=u,=n1,kssluws1:Key));
5
6   (* Send the registration request *)
7   event UserRequestsRegistration(e);
8   out(net, encrypt((msg1, u, ws, e), kssluws1));
9
10  (* Receive via SSL the SSL ticket and the combination of
11     SSL and email tickets *)
12  in(net, cmsg3:bitstring);
13  let (=msg3, =ws, =u, =e, ticketssl:Ticket, ne:Nonce) =
14      decrypt(cmsg3,kssluws1) in
```

Listing 2. User process code: initial registration request.

```
1   (* Fetch from the MS the email with the ticket sent by the WS *)
2
3   (* First: establish an SSL session with MS *)
4   new n2:Nonce;
5   out(privateSSLuserchannel, (u,ms,n2));
6   in(privateSSLuserchannel, (=ms,=u,=n2,ksslums1:Key));
7
8   (* Second: Authenticate and receive the email *)
9   out(net, encrypt((msg4,u,ms,e,pwd),ksslums1));
10  in(net, cmsg5:bitstring);
11
12  (* If the ticket received by email matches the first one (sent
13     with the SSL ticket, instruct the MS to send an ACK *)
14  let (=msg5, =ms, =u, =ne) = decrypt(cmsg5,ksslums1) in
15
16  event UserRequestsSendACK(e,ticketssl,ne);
17  out(net, encrypt((msg6,u,ms,e,ne,pwd),ksslums1));
```

Listing 3. User process code: secure email verification and acknowledgment.

```
1   (* Send both tickets back to the WS in a new SSL session *)
2   new n3:Nonce;
3   out(privateSSLuserchannel, (u,ws,n3));
4   in(privateSSLuserchannel, (=ws,=u,=n3,kssluws2:Key));
5
6   out(net, encrypt((msg8,u,ws,e,ticketssl),kssluws2));
7   in(net, cmsg9:bitstring);
8
9   let (=msg9,=ws,=u,=e,=ticketssl,id:Id) = decrypt(cmsg9,kssluws2) in
10  event UserReceivesId(e,id,ticketssl);
11  0.
```

Listing 4. User process code: identity retrieval.

```
1   let msprocess(kmsprv:PrvKey, kwspub:PubKey) =
2
3   (* Receives an email *)
4   (* Note that it is not encrypted nor authenticated *)
5   in(net, (=msg2, =ws, =ms, e:Email, ne:Nonce));
6
7   (* Receives a SSL session *)
8   in(privateSSLmschannel, (u:Host,=ms,n2:Nonce,ksslums1:Key));
9
10  (* Receive authentication via the SSL session *)
11  in(net, cmsg4:bitstring);
12  let (=msg4,=u,=ms,=e,pwd:Password) = decrypt(cmsg4, ksslums1) in
13
14  (* Check the password *)
15  get msaccounts(=e,=pwd) in
16
17  (* Send email securely via SSL *)
18  (* Note that at this point, the user is authenticated to the MS *)
19  out(net, encrypt((msg5,ms,u,ne), ksslums1));
20
21  (* Wait for an instruction to send ACK *)
22  in(net, cmsg6:bitstring);
23  let (=msg6,=u,=ms,=e,=ne,=pwd) = decrypt(cmsg6,ksslums1) in
24
25  (* Send signed ACK to WS *)
26  event MSSendsACK(e,ne);
27  out(net, ((msg7,ms,ws,e,ne), sign((msg7,ms,ws,e,ne),kmsprv)));
28  0.
```

Listing 5. User process code: identity retrieval.

and through the recently established SSL session, the email previously received. Hence, at this point the legitimate owner of the email account has received the registration email (lines 10–14 in Listing 3). When the MS receives from the user (lines 22–23 of Listing 5) the message instructing it to send the `ACK` message to the WS, prepares it and sends it along with the corresponding digital signature (in line 27).

4.4. The Web Server

For readability, we omit the WS code, since its behavior is determined by the code presented for the User and the MS (the model source code is completely available in [6]). However, it is worth noting that the WS is charge of a set of verifications that should be carried out very carefully. First, it needs to verify that the email

address belongs to a trusted domain (and that this domain provides the ACK functionality). After sending the SSL ticket and the email nonce via the SSL session, and the email to the MS, the WS will mark the status of the current registration as *mail verification pending*. It is of the utmost importance that the WS does not continue the registration process until it receives the digitally signed ACK message from the MS. After receiving the ACK message and the SSL ticket (and verifying that the ticket matches the one sent during this registration), the WS has the certainty that whoever started the registration is the owner of the email account. Thus, it can complete the registration process by creating and sending the digital identity using the same SSL session established to send back the SSL ticket.

4.5. Verification with ProVerif

For a registration protocol, we need to prove two security properties confidentiality (secrecy in ProVerif's jargon) of the conveyed identity and authenticity. More specifically, we need to prove user authenticity, which in our case also requires to prove authenticity of the MS, and authenticity of the WS. In order to prove secrecy with ProVerif, it is only necessary to specify the concrete piece of information that needs to be kept secret. This is simply done by adding a query like `query attacker(T)`, which asks if the attacker gains knowledge of T. Proving authenticity is a bit more complicated. In ProVerif an attacker is by definition a process without events, which are special instructions that mark concrete points in the execution. Since attackers cannot "launch" these events, each time one occurs, it has been executed by a legitimate process. Thus, ProVerif can be asked to prove whether or not a certain sequence of events (called correspondence assertions in ProVerif) is always satisfied. Hence, by defining an appropriate chain of events, it is possible to prove that only the intended and legitimate entities take part in the protocol.

The first query in our model proves secrecy. With the instruction.

```
query id:Id; attacker (new id).
```

We explicitly ask ProVerif to prove whether or not the attacker can gain knowledge of the distributed identities. ProVerif returns that the property is satisfied and, as a consequence of the soundness property of ProVerif, the identities issued using our protocol are confidential and not accessible to third parties.

In order to prove authenticity, we define six events and two correspondence assertions. Namely, in order of occurrence in a typical execution flow:

Event UserRequestsRegistration (Email). Executed by the User process, signals the execution point where a legitimate user requests to be registered with the specified email.

Event UserRequestsSendACK (Email, Ticket, Nonce). Invoked by the User, marks when a legitimate user has processed an email associated to the registration request with a corresponding pair of SSL ticket and email nonce.

Event MSSendsACK (Email, Nonce). Launched by the MS, signals when the MS sends a digitally signed ACK message in response to the email previously sent by the WS to the specified email address, containing the specified nonce.

Event WSProcessesACK (Email, Ticket, Nonce). This event is executed by the WS, and marks when does it receive and process an ACK message responding to the registration email sent to the specified address, containing the specified nonce, which was sent for the same registration request than the given ticket. This implies that the received tokens are valid, and the email address is now verified.

Event WSSendsId (Email, Id, Ticket). Signals the point where the WS sends a new ID to the legitimate owner of the given email address. Moreover, this ID has been acquired using the specified SSL ticket. It is launched by the WS.

Event UserReceivesId (Email, Id, Ticket). Finally executed by the User, indicates that the legitimate owner of the specified email address has received an identity after sending back the previously received SSL ticket.

Using these events, we define the correspondence assertions shown in Eqs. (1) and (2) (we drop here the syntax used by ProVerif). Even though we have eliminated it for simplicity in the equations shown here, the actual queries in the source code prove injectivity. In other words, they prove that for each of the following events on the left hand side of the $\Rightarrow$, the events on the right hand side have been executed exactly once.

$$UserReceivesId(e, id, ts) \Rightarrow (WSSendsId(e, id, ts) \wedge UserRequestsRegistration(e)). \quad (1)$$

$$WSProcessesACK(e, ts, n) \Rightarrow (MSSendsACK(e, n) \wedge UserRequestsSendACK(e, ts, n)). \quad (2)$$

Eq. (1) proves that, each time a user receives an identity, the WS has issued it and, also, the user who receives it is the user who requested it. Moreover, these events are all tied up together by the same SSL ticket (the variable name *ts*) and the email *e*. Hence, this correspondence assertion would prove that a user cannot be tricked into receiving an identity other than the one that she has requested. Finally, it also proves that when the WS issues an identity, the legitimate owner of the specified email address is the only one receiving.

The correspondence in Eq. (2) asks whether or not each time the WS processes (and accepts) an ACK message, the MS has previously sent it and the legitimate owner of the associated email account has requested the MS to send it. Moreover, all these events correspond to the same email address (*e*), the same email message (identified by the nonce *n*) and belong to the registration request in which the SSL ticket *ts* was generated.

By asking ProVerif to prove both correspondence assertions, we verify all the required authenticity properties. That is, that the legitimate owner of a concrete email account has requested to be registered, has proven to be the owner of the account by instructing the MS to send the ACK, and that she and no other entity receives the identity created during that same request. Given that ProVerif successfully proves both queries, our protocol indeed provides the mentioned authenticity guarantees.

5. A minimality analysis

In this section we study why are the main components of the protocol necessary. This includes the SSL/TLS ticket, the ACK message and the email nonce. However, we do not include here the general purpose message components like the message tags, which are used for easing the debugging process, nor the sender and recipient identities, which are a general good practice in protocol design, as stated in [23]. Additionally, we will finish this section by making some remarks on instruction *ordering* issues that might thwart the protocol's objective, lest they are executed in the wrong order. We will use the notation $enc_K(\cdot)$ when a message is sent (symmetrically) encrypted under the (symmetric) key *K*, and $sign_{priv(H)}(\cdot)$ when a message is sent signed under the private key of host *H*. For readability, we do not include the SSL/TLS key negotiations, and just use the name K_{SSL_i} for the key negotiated and used during the *i*th SSL session within an attack trace. We also omit in the attack traces shown below the message tags and the sender

and recipient identifiers that were included in the messages of the formal definition of the protocol.

5.1. SSL/TLS ticket

This element plays a key role in our protocol, since it allows us to guarantee that the user who starts a registration request will be the only one capable to complete it. This can be tested by removing the field `ticketssl` from all the messages (and events) in the source code provided in [6]. Once this so weakened model of the protocol is run with ProVerif, the (obvious) attack trace informally shown in Listing 6 is returned.

5.2. ACK message

As we have emphasized above, the `ACK` message sent to the WS is essential in our protocol. It is the message guaranteeing that the User is indeed the owner of the email address that she specified. Without it, even with the SSL ticket, an attacker would be able to obtain an identity related to any user. Again, removing the ACK message from the provided code [6], we obtain the attack trace shown in Listing 7 with ProVerif, which is precisely the attack trace corresponding to the attack in Fig. 2.

In our protocol, the email nonce is used to guarantee that the `ACK` received corresponds to a single email sent to the user, and to allow the user to check that the email received actually corresponds to the original request. Hence, it might be tempting to also use it to prove that the user has read the email (this is exactly what is done in classic EBIA). However, given that the email is sent unencrypted, an attacker could easily obtain it and send it back to the WS in the last SSL session. Thus, just using the knowledge of the email nonce as an *implicit ACK* is not enough. Indeed, this is the assumption in classic EBIA that makes the protocol insecure.

5.3. Email nonce

If the WS does not include the email nonce in the same message where the SSL ticket is sent, another attack is enabled, since the user is not able to link her legitimate registration request with the received registration email. Thus, even by sending the `ACK` message, an attack is possible. By removing the email nonce from message 3 in Fig. 3 (and thus the check in message 5 that compares the nonce received via SSL and the nonce in the email), we encounter the attack trace depicted in Listing 8, after some polishing of the trace produced by ProVerif.

Note that the WS receives two registration requests related to the same email. However, this may very well be a normal situation. For instance, a user could request registration but, somehow, lose the received SSL ticket and thus, in order to complete registration, she would need to fetch a new one. Taking advantage of this, when the attacker sees its victim requesting registration, she sends a new request for the same email address, and gets a valid ticket. The user may receive two registration emails. However, she could easily think that it is a server malfunction (after all, she has actually requested registration). Furthermore, the attacker could even block one of those two emails, in order to avoid suspicion. After the user sends the `ACK`, the attacker establishes a new SSL session with the WS and provides it the ticket obtained in the previous step. Since the WS has already received the `ACK`, it sends the identity to the attacker.

5.4. The importance of a correct ordering

From the previous subsections we can already deduce that the checks performed by the users and the WS are of the greatest importance. However, it is worth emphasizing that not only the checks, but also the order in which they are performed with respect to the messages reception and sending, is crucial for the protocol.

Firstly, the user must check that the nonce contained in the received email actually matches the nonce received altogether with the SSL ticket. Once this check is successfully performed, and only then, the user can safely instruct the MS to send the signed `ACK` to the WS. If this order is not followed, an attack similar to the one shown in Listing 8 is enabled.

Secondly, the WS needs to wait until receiving the signed `ACK` message from the MS before completing a registration request. That is, even though the WS receives back the SSL ticket from an already initiated registration request, it must not finalize it until the `ACK` has been received. As a matter of fact, the `ACK` is the only token proving that the user who is sending back the SSL ticket is the owner of the email address. If this order is broken, an attack like the one depicted in Listing 7 would be possible.

6. Usability, additional costs and trust assumptions

As we have shown, our protocol guarantees authenticity of the registering users and secrecy of the created identities. Thus, it implements a correct registration protocol. Still, it remains to analyze how usable our protocol is, and whether the introduced modifications imply acceptable costs (both in terms of communication/computational costs and in terms of the required modifications on the underlying infrastructure) or if they are otherwise unrealistic.

From the design complexity perspective, and concerning the necessary modifications to the underlying infrastructure, our protocol strictly follows the properties of the infrastructure that is currently being used by EBIA systems. Namely, we have considered the security properties provided by the available communication channels, i.e., confidential and server authenticated SSL/TLS sessions, and unencrypted and unauthenticated email messages. The only necessary addition is an authenticated channel between two trusted servers (the Mail Server and the Web Server performing the registration). However, even though this supposes an extension to the typical functionality of Mail Servers, it is not a too cumbersome functionality to be implemented. In fact, the DomainKeys Identified Mail (DKIM) [24] method provides a very close functionality. By means of DKIM, a signing domain digitally signs an email

1 $User \rightarrow WS : enc_{K_{SSL_1}}(email)$
2 $WS \rightarrow User : enc_{K_{SSL_1}}(Nonce_{email})$
3 $WS \rightarrow MS : Nonce_{email}$
4 $User \rightarrow MS : enc_{K_{SSL_2}}(email, password)$
5 $MS \rightarrow User : enc_{K_{SSL_2}}(Nonce'_{email})$
6 User: Checks $Nonce'_{email} = Nonce_{email}$
7 $User \rightarrow MS : enc_{K_{SSL_2}}(ACK)$
8 $MS \rightarrow WS : sign_{prv(MS)}(ACK, Nonce_{email})$
9 $Attacker \rightarrow WS : enc_{K_{SSL_3}}(email)$
10 $WS \rightarrow Attacker : enc_{K_{SSL_3}}(ID(email))$

Listing 6. Attack trace when the SSL ticket is not used. The attacker might wait until a legitimate user starts a registration and instructs the MS to send the `ACK` message. Afterwards, blocking the user's communications and establishing an "illegitimate" SSL session with the WS she would be able to retrieve the user's identity.

$$
\begin{array}{ll}
1 & Attacker \rightarrow WS : enc_{K_{SSL_1}}(email_{victim}) \\
2 & WS \rightarrow Attacker : enc_{K_{SSL_1}}(Ticket_{SSL}, Nonce_{email_{victim}}) \\
3 & WS \rightarrow MS : Nonce_{email_{victim}} \\
4 & Attacker \rightarrow WS : enc_{K_{SSL_2}}(Ticket_{SSL}, email_{victim}) \\
5 & WS \rightarrow Attacker : enc_{K_{SSL_2}}(ID(email_{victim}))
\end{array}
$$

Listing 7. Attack trace when the ACK is removed from the protocol. The attacker starts from the beginning a new registration process, specifying the email address of its victim.

$$
\begin{array}{ll}
1 & User \rightarrow WS : enc_{K_{SSL_1}}(email) \\
2 & WS \rightarrow User : enc_{K_{SSL_1}}(Ticket_{SSL}) \\
3 & WS \rightarrow MS : Nonce_{email} \\
4 & Attacker \rightarrow WS : enc_{K_{SSL_2}}(email) \\
5 & WS \rightarrow Attacker : enc_{K_{SSL_2}}(Ticket'_{SSL}) \\
6 & WS \rightarrow MS : Nonce'_{email} \\
7 & User \rightarrow MS : enc_{K_{SSL_3}}(email, password) \\
8 & MS \rightarrow User : enc_{K_{SSL_3}}(Nonce'_{email}) \\
9 & User \rightarrow MS : enc_{K_{SSL_3}}(ACK(Nonce'_{email})) \\
10 & MS \rightarrow WS : sign_{prv(MS)}(ACK, Nonce'_{email}) \\
11 & Attacker \rightarrow WS : enc_{K_{SSL_4}}(Ticket'_{SSL}, email) \\
12 & WS \rightarrow Attacker : enc_{K_{SSL_4}}(ID(email))
\end{array}
$$

Listing 8. Attack trace when the email nonce is not sent jointly with the SSL ticket. The attacker can make the WS believe that it is issuing an ID related to an acknowledged email account, to the same user who requested to be registered.

Table 1
Relations of messages present in our proposal and present/absent in EBIA or [5]. The numbers shown in the first row represent the associated message of our protocol as shown in Fig. 3 (e.g., message 2 corresponds to the message where $ticket_{SSL}$ and $nonce_{email}$ are sent). In the column below each number, we write "NO" when the corresponding message is not present in EBIA/[5] and "YES" if it is indeed present. Note that messages 3–5 actually belong to the process of sending an email and fetching it from the Mail Server, hence, they are always present.

Scheme	1	2	3	4	5	7	8	9	11
EBIA	YES	NO	YES	YES	YES	NO	NO	YES*	YES
[5]	YES	YES	YES	YES	YES	NO	NO	YES	YES

* In EBIA, $nonce_{email}$ is sent here.

message, gaining some responsibility over it. Many email providers (e.g. Gmail, Yahoo) use it to reduce spam by signaling *trusted* emails Also, according to http://eggert.org/meter/dkim, it is a widely deployed technology. For instance, an email provider could easily implement our ACK method by adding a few email filtering rules in conjunction with DKIM. From the point of view of the communication costs, we show in Table 1 a relation of which messages are added with respect to EBIA and with respect to the equivalent solution of [5]. We compare with [5] because it is the one that most resembles our proposal in terms of aims and construction. Note that three messages are added when compared to EBIA, and only two when comparing with [5]. With respect to EBIA, we add the SSL ticket, the nonce sent in the second message of our protocol, and the messages used as explicit acknowledgment (originated by the user and forwarded by the Mail Server). In [5] the message containing the SSL ticket is already sent, so the only addition is the set of messages involved in the acknowledgment. Moreover, each of these extra messages would probably be just a few tens of bytes, hence they do not imply an unacceptable communication overcost. As for computational costs, compared to EBIA, our scheme adds the computation of an extra random element (the SSL ticket) by the Web Server, the creation of a digital signature by the Mail Server (the ACK), and two bitstring comparisons (steps 6 and 10 in Fig. 3); compared to [5], our proposal just adds the computation of the digital signature and the bitstring comparison in step 6 of Fig. 3. The bitstring comparisons are cheap operations, and there are very efficient implementations of (pseudo) random number generation and digital signatures. Besides, considering that registrations are not the most frequent action in web systems, the extra costs would probably be quite acceptable.

It is also important for the feasibility of our protocol to ponder if the imposed trust relationships are realistic. And again, they are quite bearable. Note that we are using email addresses as primary identifiers for the users that will be registered with our protocol. Hence, we need to trust that, when an account is compromised, the user will take the necessary measures to, at least, inform of this fact. Thus, this trust relation depends directly on the measures that the email provider and its users take to protect accesses to their email accounts. Luckily, it is a common practice nowadays to protect authentication to Mail Servers using SSL/TLS and other additional security procedures, like the Google Authenticator [25]. Note also that our protocol could be extended with additional multichannel techniques during registration, like SMS, to enable verification of mobile phone numbers (although guaranteeing that the owner of the specified mobile phone number and email address are the same might be more challenging than what it seems at first sight). Concerning the servers in our protocol (i.e., the Mail Server and the Web Server), they are assumed to be trusted entities which have publicly trusted digital identities. Thus, establishing mutually authenticated sessions between them, and server-side authenticated sessions with any other entity, is something trivial. Of course, we do not take into account attacks in which the user ignores server side certificates, or is deceived by seemingly legitimate, but illegitimate, ones. Therefore, it seems that all the requirements placed by our protocol, both related to technical and trust matters, are reasonable given current infrastructures.

Finally, from the usability perspective, everything could be done just like with classic EBIA. Following the approach taken in [5], a plugin for email clients could be developed to perform all the user-side tasks automatically. That is, the plugin will receive and store the SSL ticket and the expected email nonce. Once the corresponding email has been received, it would check that the nonce included in the incoming email actually matches the nonce received via SSL/TLS. Afterwards, the instructs the Mail Server to send the authenticated ACK to the Web Server. As a matter of fact, a preliminary version of this protocol was actually implemented and applied to the Moodle platform [26] (this preliminary version did not include the explicit authentication message although, as we have observed, this could be done without affecting usability). Moreover, we conducted a survey on its usability and concluded that it was an acceptably usable proposal (see [27, Section 5.5] for a summary of the usability test). Hence, our proposal achieves a high level of security suitable for many situations without eroding usability.

7. Conclusion

The development of strategies to promote the synergy between theory-based and practice-based approaches is the cornerstone of modern proposals for preserving information security. One possible solution to fill the gap between theoretical assumptions and the application context consists of adopting standard protocols, and adapting them into the specifics of practical scenarios. Moreover, both the selection of standards and the further adjustment must be evaluated and validated through a rigorous procedure. In this work we have addressed this challenge for the specific case of registration protocols for the exchange of digital identities. On one hand, the theoretical part of our analysis is based on the Dolev–Yao model and the correctness property of ProVerif. On the other hand, the practice oriented counterpart is determined by the commitments of usability and the adoption of standard solutions. Thus, we have improved the user-friendly EBIA registration protocol by means of SSL/TLS, DKIM and standard challenge/response cryptographic primitives. The validation of the new EBIA-based registration protocol has been achieved by analyzing its security properties using the theoretical attacker model determined by the Dolev–Yao paradigm, and implementing that evaluation model according to the application context with the help of the automatic protocol verifier ProVerif. In short, we have demonstrated that the model of our protocol satisfies confidentiality of the created identities, and authenticity of the conveyed information and the participating entities.

Our proposal guarantees the authenticity property necessary for implementing secure registration protocols, by means of adding to EBIA an extra authenticated message and a token sent via SSL. Specifically, the authenticated message is sent by a Mail Server, belonging to the domain where the user owns an email account; and the SSL token is sent during the first interaction between the registering user and the Web Server. Moreover, we have demonstrated, with the help of ProVerif, that all the elements in our protocol are necessary, since eliminating any of them enables attacks on it. Furthermore, the order in which each action is performed is also crucial for the same reason.

We have also considered the additional communication and computational costs induced by our protocol, which seem acceptable; the required modifications to the existing infrastructures are basically unnecessary or minimal, since there already exist protocols (concretely, we have underlined the suitability of the DomainKeys Identified Mail protocol) that are suitable for our requirements; and the extra necessary trust relationships, which are quite bearable given that Mail Servers usually own digital identities for authenticating themselves. Finally, from the usability perspective, we have shown that our protocol may be implemented in such a manner that the final user does not perceive any difference between it and classic EBIA. Moreover, it could be easily adapted to distribute any kind of digital identity, ranging from the typical username and password pairs to the most advanced anonymous digital X.509 identities [28,29]. In fact, once the connection between the user and the Web Server is established, the Web Server has the certainty that the user is the real owner of the provided email address. As a consequence, the Web Server can assume that there exists an adequate channel to send the generated digital identity back to the user.

Acknowledgments

The authors are very grateful to the reviewers for their constructive comments. This work was supported by the UAM project of Teaching Innovation EPS-L1.2.13 and the Spanish Government projects TIN2010-19607 and TIN2012-30883. The work of David Arroyo was supported by a Juan de la Cierva fellowship from the Ministerio de Ciencia e Innovaciòn of Spain.

References

[1] S.L. Garfinkel, Email-based identification and authentication: an alternative to PKI?, IEEE Secur Privacy 1 (6) (2003) 20–26.
[2] S.L. Garfinkel, D. Margrave, J.I. Schiller, E. Nordlander, R.C. Miller, How to make secure email easier to use, in: CHI, 2005, pp. 701–710.
[3] B. Blanchet, ProVerif automatic cryptographic protocol verifier user manual, CNRS, Département d'Informatique École Normale Supérieure, Paris, 2010.
[4] F.-L. Wong, F. Stajano, Multi-channel protocols, in: Security Protocols Workshop, 2005, pp. 112–127.
[5] T.W. van der Horst, K.E. Seamons, Simple authentication for the web, in: SecureComm, 2007, pp. 473–482.
[6] J. Diaz, D. Arroyo, F. Rodriguez, ProVerif model for the protocols studied in the present work, 2012a. <http://www.ii.uam.es/~gnb/registration-ack.tgz>.
[7] B. Adida, Beamauth: two-factor web authentication with a bookmark, in: ACM Conference on Computer and Communications Security, 2007, pp. 48–57.
[8] A. Whitten, J.D. Tygar, Why Johnny can't encrypt: a usability evaluation of PGP 5.0, in: Proceedings of the 8th Conference on USENIX Security Symposium, vol. 8, SSYM'99, USENIX Association, Berkeley, CA, USA, 1999, pp. 14–14. <http://dl.acm.org/citation.cfm?id=1251421.1251435>.
[9] S.L. Garfinkel, R.C. Miller, Johnny 2: a user test of key continuity management with S/MIME and outlook express, in: SOUPS, 2005, pp. 13–24.
[10] S. Sheng, L. Broderick, C. Koranda, J. Hyland, Why Johnny still can't encrypt: evaluating the usability of email encryption software, in: L.F. Cranor (Ed.) Proceedings of the 2nd Symposium on Usable Privacy and Security, SOUPS 2006, Pittsburgh, Pennsylvania, USA, July 12–14, 2006. ACM 2006 ACM International Conference Proceeding Series, ISBN 1-59593-448-0.
[11] W. Enck, P. Traynor, P. McDaniel, T.F.L. Porta, Exploiting open functionality in SMS-capable cellular networks, in: ACM Conference on Computer and Communications Security, 2005, pp. 393–404.
[12] C. Newman, Using TLS with IMAP, POP3 and ACAP, 1999. <http://tools.ietf.org/html/rfc2595>.
[13] M. Burrows, M. Abadi, R.M. Needham, A logic of authentication, ACM Trans. Comput. Syst. 8 (1) (1990) 18–36.
[14] W. Mao, C. Boyd, Towards formal analysis of security protocols, in: CSFW, 1993, pp. 147–158.
[15] M. Abadi, P. Rogaway, Reconciling two views of cryptography (the computational soundness of formal encryption), in: IFIP TCS, 2000, pp. 3–22.
[16] A. Horn, On sentences which are true of direct unions of algebras, J. Symb. Log. 16 (1) (1951) 14–21.
[17] B. Blanchet, An efficient cryptographic protocol verifier based on prolog rules, in: CSFW, 2001, pp. 82–96.
[18] B. Blanchet, From secrecy to authenticity in security protocols, in: SAS, 2002, pp. 342–359.
[19] R. Milner, J. Parrow, D. Walker, A Calculus of Mobile Processes, I, Inf. Comput. 100 (1) (1992) 1–40.
[20] M. Abadi, A.D. Gordon, A calculus for cryptographic protocols: the spi calculus, in: ACM Conference on Computer and Communications Security, 1997, pp. 36–47.
[21] D. Dolev, A.C.-C. Yao, On the security of public key protocols, IEEE Trans. Inform. Theory 29 (2) (1983) 198–207.
[22] J. Diaz, D. Arroyo, F.B. Rodriguez, A formal methodology for integral security design and verification of network protocols, J. Syst. Softw. 89 (0) (2014) 87–98, http://dx.doi.org/10.1016/j.jss.2013.09.020.
[23] M. Abadi, R.M. Needham, Prudent engineering practice for cryptographic protocols, IEEE Trans. Softw. Eng. 22 (1) (1996) 6–15.
[24] D. Crocker, T. Hansen, M. Kucherawy, DomainKeys Identified Mail (DKIM) Signatures, RFC 6376 (Draft Standard), 2011. <http://datatracker.ietf.org/doc/rfc6376/>.
[25] Google, google-authenticator, 2013. <http://code.google.com/p/google-authenticator/>.
[26] J. Diaz, D. Arroyo, F.B. Rodriguez, An approach for adapting moodle into a secure infrastructure, in: CISIS, 2011, pp. 214–221.
[27] J. Diaz, D. Arroyo, F.B. Rodriguez, Formal Security Analysis of Registration Protocols for Interactive Systems: A Methodology and A Case of Study, CoRR abs/1201.1134.
[28] V. Benjumea, S.G. Choi, J. Lopez, M. Yung, Anonymity 2.0 – X.509 extensions supporting privacy-friendly authentication, in: CANS, 2007, pp. 265–281.
[29] J. Diaz, D. Arroyo, F.B. Rodriguez, Anonymity revocation through standard infrastructures, in: EuroPKI, 2012c, pp. 112–127.