Efficient Multi-Party Challenge-Response Protocols for Entity Authentication

Levente Buttyán,[*] Attila Nagy,[†] István Vajda[‡]

April 2001

[*]Swiss Federal Institute of Technology – Lausanne, Institute for Computer Communications and Applications, EPFL-DSC-ICA, CH-1015 Lausanne, levente.buttyan@epfl.ch

[†]Budapest University of Technology and Economics, Faculty of Natural Sciences, nagy@math.bme.hu

[‡]Budapest University of Technology and Economics, Department of Telecommunications, Sztoczek u. 2, H-1111 Budapest, vajda@hit.bme.hu

Abstract

In this paper, we address the problem of multi-party entity authentication. We prove that the lower bound on the number of messages of multi-party challenge-response protocols is $2n-1$, where n is the number of the participants of the protocol, and propose two protocols that achieve this lower bound. Our protocols are, thus, efficient in the sense that they use the minimum number of messages required to solve the multi-party entity authentication problem based on challenge-response principles.

keywords: challenge-response protocols, protocol graph, entity authentication, reflection attack

1 Introduction

Entity authentication is the process whereby a party gains assurance of the identity of another party involved in a protocol [7]. Entity authentication is a fundamental security service, which is used for preventing impersonation and unauthorized access to services in distributed systems. Common examples for entity authentication include user authentication in computer systems (login procedure) and subscriber authentication in GSM networks.

Strong entity authentication is based on cryptographic challenge-response protocols, in which a party (the prover) proves its identity to another party (the verifier) by demonstrating knowledge of a secret that is known to be associated with the prover. This is done by providing a response to a time-variant challenge, where the response depends on both the secret and the challenge in such a way that an attacker cannot obtain the secret from the response. Furthermore, since subsequent challenges differ, the attacker cannot use the response from one execution of the protocol in a subsequent execution. Depending on the mechanisms used, the verifier may or may not know the secret that is used in the computation of the response. If the verifier does not know the secret, nevertheless, it can still verify the response, then the protocol is called zero-knowledge protocol [10]. In this paper, we are not concerned with this type of protocols, but exclusively focus on classical challenge-response protocols, where the verifier knows the secret associated with the prover, and uses it to verify the response.

A considerable amount of work has been carried out on the design and analysis of two-party challenge-response protocols for entity authentication [9, 3]. In this paper, we consider the multi-party case, which, to the best of our knowledge, has been neglected so far. In multi-party entity authentication, each of the n $(n \geq 2)$ participating parties proves its identity to each of the other parties. Although, in principle, multi-party entity authentication can be obtained by running two-party mutual entity authentication protocols between each pair of parties, in practice, this approach is not desirable, because it leads to highly inefficient protocols that use $O(n^2)$ messages. We propose much more efficient protocols that use only $O(n)$ messages. Furthermore, we show that our protocols are optimal in the sense that no protocol can solve the problem with less number of messages than ours do.

In spite of their apparent simplicity, the design of entity authentication protocols is surprisingly error prone, especially, if they are combined with session key establishment. Many protocols have been proposed that were found to be flawed and vulnerable to some form of replay attack later [5]. The reason for this is that flaws are usually subtle and hard to find. In order to solve this problem, many papers propose methods that can be used for formal verification of entity authentication and key establishment protocols [4, 6, 11], and principles that can help to avoid common mistakes in their design [1, 2]. In this paper, we do not aim at contributing to these efforts, but we rather build on them: we adhere to the design principles of [1] and use a formal logic [11] to explain some of the subtle details of our protocols.

The outline of the paper is the following. In Section 2, we introduce our system model and clarify the concept of entity authentication in this model. Then, in Section 3, we prove that the lower bound on the number of messages of multi-party challenge-response protocols for entity authentication is $2n-1$, where n is the number of participants of the protocol. Before presenting our protocols, which achieve this lower bound, we review two flawed entity authentication protocols in Section 4. Our aim is to give an insight into two design principles that our protocols build on. In Section 5, we present our protocols and analyze them with the help of a formal logic. Finally, in Section 6, we conclude the paper.

2 System model and the goal of entity authentication

We consider a system that consists of a set of principals (users, hosts, and processes) and a network that connects them. Principals communicate with each other by sending messages via the network. In order to authenticate each other, a subset of the principals may engage in a given multi-party entity authentication protocol. We assume that all the principals know this protocol. We also assume that any of the principals can play any of the roles in the protocol[1] (e.g., in case of two-party protocols, anybody can be initiator as well as responder). We further assume that principals may run several instances of the protocol concurrently, and play different roles in different instances.

As usual in the literature [8], we assume that the network is under the control of the attacker. This means that the attacker can observe every message sent via the network, furthermore, it can intercept, modify, generate, delay, and replay messages or parts of them. We assume that the attacker knows the protocol that is run by the principals, and it may try to play any of its roles. In addition, it can arrange that a principal starts an instance of

[1]Later, we will introduce special roles (e.g., authentication server), which can be played only by designated principals. We omit this issue in the presentation of the general system model, because it depends on the particular protocol in question.

the protocol at any time chosen by the attacker. On the other hand, the attacker does not know any of the long-term secrets associated with legitimate principals (see also next paragraph), and it cannot break the cryptographic primitives used for encryption, digital signature, etc. This leaves the attacker with the only possibility to mount a replay attack, in which it tries to impersonate some principals by constructing fake messages from data recorded in previous and/or concurrent runs of the protocol.

Sometimes we assume that the attacker compromised the long-term secret of a principal or a small subset of principals. In this case, we are interested in if the attacker can use the compromised secret(s) to impersonate a principal that is not compromised. If the authentication protocol is designed properly, then this should not be possible. Note, however, that the attacker can always impersonate the compromised principals, no matter how careful the design of the authentication protocol was.

As we said before, entity authentication is the process whereby a party gains assurance of the identity of another party involved in a protocol. At first sight, this suggests that a principal can use an entity authentication protocol to verify that the identity of another principal with which it communicates (i.e., from which it received a message), is as claimed. Note, however, that in our system model, each principal does actually communicate with the attacker, because messages are sent to and received from the network, which is under the control of the attacker. What can an entity authentication protocol achieve in this model? Indeed, all we can expect from a correct entity authentication protocol is that it guarantees for a principal who successfully run it that the assumed other participating principals were present and sent some messages during the protocol run. We formalize this concept in the following definition:

Definition 1 (Entity authentication) *Let us consider two principals A and B. We say that A* authenticated *B if there exists a bounded time interval I in the local time of A such that A is convinced that B was alive (i.e., sent some messages) in I.*

Example 1 As an example let us consider the following unilateral two-party entity authentication protocol:

$$1. \quad B \rightarrow A: \quad \{T\}_{K_b^-}$$

The protocol works as follows: B digitally signs the current value T of its local clock using its private key K_b^-, and sends the signed time-stamp

$\{T\}_{K_b^-}$ to A. It is assumed that the clocks of A and B are synchronized with some accuracy Δt. This means that at any time t the local clock $c_a(t)$ of A and the local clock $c_b(t)$ of B do not differ more than Δt (i.e., $\forall t : |c_a(t) - c_b(t)| \leq \Delta t$). When A receives the message, it verifies the digital signature of B. If this verification is successful, then A authenticated B, since it is convinced that B was alive and used its private key at some time in the interval $[T - \Delta t, T + \Delta t]$ in the local time of A (see Figure 1). □

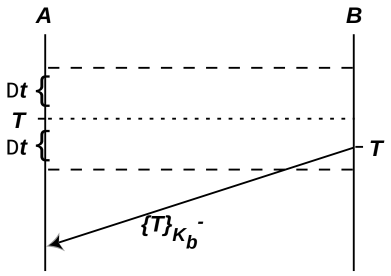

Figure 1: A is convinced that B was alive at some time in the interval $[T - \Delta t, T + \Delta t]$

Example 2 Another common example for a unilateral two-party entity authentication protocol is the following:

$$\begin{array}{lll} 1. & A \to B: & r \\ 2. & B \to A: & \{r\}_{K_b^-} \end{array}$$

Here, A generates an unpredictable random number r, and sends it to B at time T_1 in its local time. B signs r with its private key K_b^-, and sends the result $\{r\}_{K_b^-}$ back to A. A receives B's response at time T_2 in its local time, and verifies that it is indeed its random number r signed by B. If this verification is successful, then A authenticated B, since it is convinced that B was alive and used its private key at some time in the interval $[T_1, T_2]$ (see Figure 2). □

3 Lower bound on the number of messages

After having defined what we mean by entity authentication, we now turn our attention to multi-party entity authentication protocols in which each

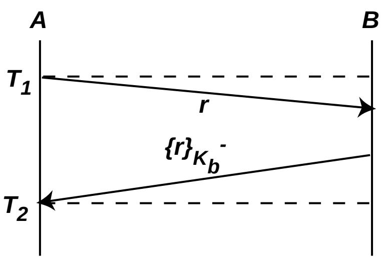

Figure 2: A is convinced that B was alive at some time in the interval $[T_1, T_2]$

party authenticates every other participating party. We are exclusively concerned with challenge-response type protocols, where authentication is based on response to an unpredictable random challenge (like in Example 2). The question we investigate in this section is: What is the lower bound on the number of messages in multi-party challenge-response protocols for entity authentication?

We start by constructing a model of the protocol, in which we abstract away from the exact content of messages and retain only the message passing structure of the protocol:

Definition 2 (Protocol graph) *Let us represent a protocol with a directed graph $\mathcal{G} = (V, E)$, where V is the set of vertices and E is the set of edges in $\mathcal{G}$. Each vertex of $\mathcal{G}$ represents a party of the protocol, and it is labeled with the name of that party. The edges of $\mathcal{G}$ correspond to the messages of the protocol; each messages sent by party A to party B is represented by a directed edge from the vertex that is labeled with A to the vertex that is labeled with B.*

We define the following binary relations on the edges of $\mathcal{G}$:

Definition 3 (Precedence) *The precedence relation is a subset P of $E \times E$ such that for all $(e, f) \in P$ the message that corresponds to e is sent earlier than the message that corresponds to f in every execution of the protocol. If $(e, f) \in P$, then we say that e* precedes *f or f* succeeds *e, and we denote this by $e \prec f$.*

Definition 4 (Precedence or equality) *The precedence or equality relation is a subset P' of $E \times E$ defined as $P' = P \cup \{(e, f) \in E \times E : e = f\}$. We use the $e \preceq f$ notation to denote that $(e, f) \in P'$.*

It is clear that if message e is always sent earlier than message f, and f is always sent earlier then g, then e is always sent earlier than g, which means that P and P' are transitive (i.e., $e \prec f \prec g$ implies $e \prec g$, and $e \preceq f \preceq g$ implies $e \preceq g$). In addition, P' is reflexive and antisymmetric as well (i.e., $e \preceq e$, and $e \preceq f$ and $f \preceq e$ implies $e = f$). Therefore, P' is a partial ordering. The reason for being only partial and not total ordering is that the protocol may have concurrent messages, the order of which cannot be guaranteed. This means that $\mathcal{G}$ may have two edges e and f, such that neither $e \preceq f$ nor $f \preceq e$.

We find it convenient in explaining the theory to introduce a notation for directly preceding edges:

Definition 5 (Direct precedence) *An edge e* directly precedes *an edge f, denoted by $e \ll f$, if $e \prec f$ and there is no other edge g such that $e \prec g \prec f$.*

The following lemma states that directly preceding edges must be joined by a common vertex:

Lemma 1 *Let us consider a protocol graph $\mathcal{G}$. If for two edges $e = (u, v)$ and $f = (w, z)$ in $\mathcal{G}$, $e \ll f$, then $v = w$.*

Proof: Let us assume that $v \neq w$. This means that they correspond to different parties of the protocol. Let the parties that belong to v and w be A and B, respectively. In order to guarantee that message f is sent after message e in every execution of the protocol, A and B must be synchronized: A must be able to notify B that e arrived, and B must send f only if it received this notification. This means, however, that the protocol must have a message g (the notification), which succeeds e and precedes f. This contradicts our assumption that $e \ll f$. $\Box$

Lemma 2 *Let us consider a protocol graph $\mathcal{G}$. If for two edges e and f in $\mathcal{G}$, $e \prec f$, then either $e \ll f$, or there is a sequence of edges $g_1, g_2, \ldots, g_k$, where $k \geq 1$, such that $e \ll g_1 \ll g_2 \ll \ldots \ll g_k \ll f$.*

Proof: Let us denote the set of edges that succeeds e and precedes f by G (i.e., $G = \{g \in E : e \prec g \prec f\}$). If G is empty, then $e \ll f$ by definition. So let us assume that G is not empty. Let g be (one of) the "latest" edge(s) in G (i.e., there is no $g' \in G$ such that $g \prec g'$). Note that because of the finite size of $\mathcal{G}$, and thus G, such an edge always exists. g must directly precede f, because if there was an edge g' such that $g \prec g' \prec f$, then g' would be in G, and g would not be (one of) the latest edge(s). Thus, we have that

$e \prec g \ll f$. Now we can repeat the same argument for $e \prec g$. Since G is finite, after a finite number k of repetition, we are done. $\Box$

According to Definition 1, a party A authenticated a party B if A is convinced that B was alive and sent some messages in a bounded time interval I in the local time of A. In case of challenge-response protocols, I is defined by the time of sending a challenge and the time of receiving a response. Therefore, the following lemma holds for any challenge-response protocol for entity authentication:

Lemma 3 *Let us consider a challenge-response protocol for entity authentication and its protocol graph $\mathcal{G}$. Let A and B be two parties of the protocol, and let us denote the vertices that correspond to A and B by u and v, respectively. If party A authenticates party B in the protocol, then there exist three edges e, e', and f in $\mathcal{G}$ such that e is an outgoing edge from u (challenge), e' is an incoming edge to u, f is an outgoing edge from v (response), and $e \prec f \preceq e'$.*

Corollary of Lemma 3: A direct consequence of the previous lemma is that if each party authenticates at least one other party in the protocol, then each vertex of $\mathcal{G}$ has an outgoing edge e and an incoming edge e' such that $e \prec e'$. $\Box$

Lemma 4 *Let us consider a challenge-response protocol for entity authentication and its protocol graph $\mathcal{G}$. If each party authenticates every other party in the protocol, then any two vertices of $\mathcal{G}$ are connected with a directed path.*

Proof: Let us consider two vertices u and v of $\mathcal{G}$, where u corresponds to party A and v corresponds to party B. Because of Lemma 3, there exist two edges e and f such that e originates from u, f originates from v, and $e \prec f$. Using Lemma 2, we get that either $e \ll f$, or there is a sequence of edges $g_1, g_2, \ldots, g_k, (k \geq 1)$ such that $e \ll g_1 \ll g_2 \ll \ldots \ll g_k \ll f$. Because of Lemma 1, this means, in both cases, that there is a directed path from u to v. $\Box$

Corollary of Lemma 4: A consequence of Lemma 4 is that if each party authenticates every other party in the protocol, then the protocol graph is connected.$\Box$

We now introduce the notion of unfolded protocol graphs. The unfolded protocol graph $\tilde{\mathcal{G}}$ of the protocol graph $\mathcal{G}$ can be obtained by the following procedure:

We build up $\tilde{\mathcal{G}}$ from $\mathcal{G}$ step-by-step starting from an empty graph and extending it with one new edge taken from $\mathcal{G}$ in each step. During the construction of $\tilde{\mathcal{G}}$, we execute a depth-first search on the edges of $\mathcal{G}$ following the direct precedence relation on the edges. This search determines the order in which the edges of $\mathcal{G}$ are processed and inserted in $\tilde{\mathcal{G}}$, as well as the originating vertex of each new edge in $\tilde{\mathcal{G}}$.

Let us assume that the first edge given by the depth-first search is $e = (u, v)$. Since at this point $\tilde{\mathcal{G}}$ is empty, we simply insert a new edge $\tilde{e}$ (with new originating and destination vertices) in $\tilde{\mathcal{G}}$. The originating and destination vertices of $\tilde{e}$ get the same labels as u and v, respectively.

Now, let us assume that we have processed edge e' from $\mathcal{G}$ and inserted $\tilde{e}'$ in $\tilde{\mathcal{G}}$. Furthermore, let us assume that the next edge given by the depth-first search is $e'' = (u'', v'')$. There are two cases: (1) $e' \ll e''$ or (2) there is no edge that succeeds e', and e'' is obtained by *backtracking* (i.e., stepping back on already processed edges up to an edge which has an as yet unprocessed direct successor). The originating and destination vertices of the new edge $\tilde{e}''$ inserted in $\tilde{\mathcal{G}}$ is determined as follows:

Case (1)

- Originating vertex: the originating vertex of $\tilde{e}''$ is the destination vertex of $\tilde{e}'$.
- Destination vertex:
  - if a direct successor f of e'' has already been processed and the corresponding edge $\tilde{f}$ has already been inserted in $\tilde{\mathcal{G}}$, then the destination vertex of $\tilde{e}''$ is the originating vertex of $\tilde{f}$,
  - if no direct successor of e'' has been processed yet, then the destination vertex of $\tilde{e}''$ can be any vertex in $\tilde{\mathcal{G}}$ that has the same label as v'' has, given that this does not cause a directed loop in $\tilde{\mathcal{G}}$,
  - otherwise a new vertex is inserted in $\tilde{\mathcal{G}}$ with the same label as v'' has, and this new vertex becomes the destination vertex of $\tilde{e}''$.

Case (2)

- Originating vertex: we perform a backtracking in $\tilde{\mathcal{G}}$ parallel with the backtracking in $\mathcal{G}$. The vertex, in which this parallel backtracking stops, becomes the originating vertex of $\tilde{e}''$.

- Destination vertex: the same applies as in case (1).

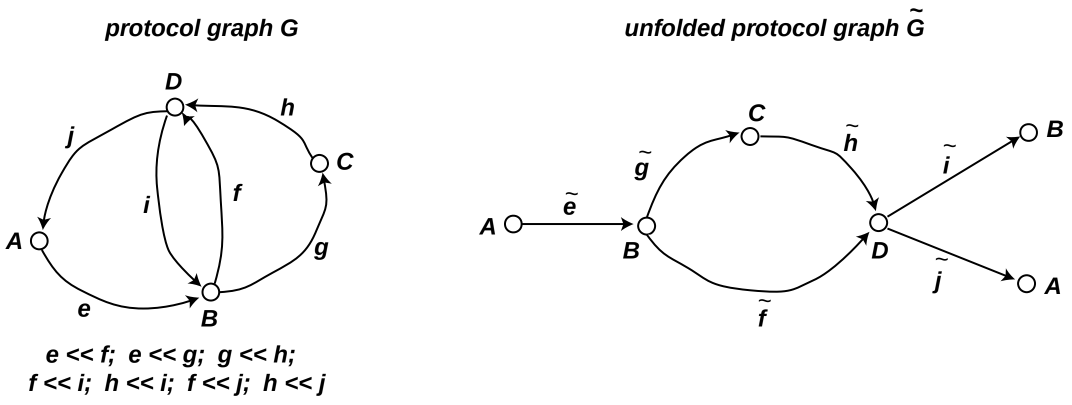

Figure 3: An example for unfolding a protocol graph

As an example, let us consider Figure 3. Given the protocol graph on the left hand side of the figure, the procedure builds the unfolded protocol graph on the right hand side. According to the depth-first search, the edges are processed in the following order: e, g, h, j, i, and f. $\tilde{e}$, $\tilde{g}$, and $\tilde{h}$ are simply inserted in $\tilde{\mathcal{G}}$ one after the other. In order to avoid a directed loop, when $\tilde{j}$ is inserted, we need to add a new vertex with label A to $\tilde{\mathcal{G}}$, and this new vertex becomes the destination vertex of $\tilde{j}$. Since j does not have any successor in $\mathcal{G}$, we then perform a backtracking, which stops at the destination vertex of h in $\mathcal{G}$, and the destination vertex of $\tilde{h}$ in $\tilde{\mathcal{G}}$. Therefore, the originating vertex of the next edge to insert (i.e., $\tilde{i}$) will be the destination vertex of $\tilde{h}$. As before, in order to avoid a directed loop, we need to add a new vertex, this time with label B, to $\tilde{\mathcal{G}}$, and this new vertex becomes the destination vertex of $\tilde{i}$. Then we perform a backtracking again, which stops at the destination vertex of e in $\mathcal{G}$, and the destination vertex of $\tilde{e}$ in $\tilde{\mathcal{G}}$. Therefore, the originating vertex of the last edge $\tilde{f}$ will be the destination vertex of $\tilde{e}$. f has two direct successors i and j in $\mathcal{G}$, and both of them have already been processed and inserted in $\tilde{\mathcal{G}}$. Thus, the destination vertex of $\tilde{f}$ will be the originating vertex of $\tilde{i}$ and $\tilde{j}$.

The following lemma guarantees that, for all the direct successors $f, f', \ldots$ of an edge e, the corresponding edges $\tilde{f}, \tilde{f}', \ldots$ originate from the same vertex in $\tilde{\mathcal{G}}$. This ensures that we can always unambiguously determine the destination vertex of an edge to be inserted in $\tilde{\mathcal{G}}$ if its direct successors have already been processed and inserted in $\tilde{\mathcal{G}}$.

Lemma 5 *If two concurrent edges e and f originate from the same vertex in $\mathcal{G}$, then the corresponding edges $\tilde{e}$ and $\tilde{f}$ originate from the same vertex in $\tilde{\mathcal{G}}$.*

Proof: Let us assume that the procedure processes e and inserts $\tilde{e}$ first. Then, it continues with the successors of e. When all the successors of e are processed we perform the backtracking in the protocol graph and in the (partial) unfolded protocol graph as well. Since f is not processed yet the backtracking stops at the originating vertex of f (which is also the originating vertex of e) in $\mathcal{G}$ and in the originating vertex of $\tilde{e}$ in $\tilde{\mathcal{G}}$. Therefore, this vertex (the originating vertex of $\tilde{e}$) will be the originating vertex of $\tilde{f}$. $\square$

The following statements are direct consequences of the unfolding procedure given above:

Lemma 6 *Let us consider a protocol graph $\mathcal{G} = (E, V)$ and its unfolded protocol graph $\tilde{\mathcal{G}} = (\tilde{E}, \tilde{V})$.*

- *$\tilde{\mathcal{G}}$ is a Directed Acyclic Graph (DAG);*
- *if $\mathcal{G}$ is connected, then $\tilde{\mathcal{G}}$ is connected as well;*
- *$|E| = |\tilde{E}|$ and there exists a one-to-one mapping $m : E \to \tilde{E}$ such that if $e \ll f$ in $\mathcal{G}$, then the destination vertex of $m(e) = \tilde{e}$ and the originating vertex of $m(f) = \tilde{f}$ are the same in $\tilde{\mathcal{G}}$.*
- *The vertices of $\tilde{\mathcal{G}}$ are labeled with the names of the protocol participants in such a way that for any edge $\tilde{e}$ in $\tilde{\mathcal{G}}$, the labels on the originating and destination vertices of $\tilde{e}$ are the same as the labels on the originating and destination vertices of $m^{-1}(\tilde{e}) = e$ in $\mathcal{G}$, respectively.*

Now, we are ready to state and prove the main result of this section:

Theorem 1 *Any n-party challenge-response protocol for entity authentication, in which each party authenticates every other party, uses at least $2n-1$ messages.*

Proof: Let us consider the protocol graph $\mathcal{G}$ of the protocol and the unfolded protocol graph $\tilde{\mathcal{G}}$. First, using the corollary of Lemma 4, we get that $\mathcal{G}$ is connected, and from this, using Lemma 6, we get that $\tilde{\mathcal{G}}$ is connected as well. Second, from the corollary of Lemma 3, we get that each vertex u of

$\mathcal{G}$ has an outgoing edge e and an incoming edge e' such that $e \prec e'$. The corresponding edges in $\tilde{\mathcal{G}}$ are $\tilde{e} = m(e)$ and $\tilde{e}' = m(e')$, respectively. The originating vertex $\tilde{u}$ of $\tilde{e}$ and the destination vertex $\tilde{v}'$ of $\tilde{e}'$ have the same labels in $\tilde{\mathcal{G}}$, because they both inherited the label of u in $\mathcal{G}$. However, $\tilde{u}$ cannot be the same as $\tilde{v}'$, since according to Lemma 2 and the construction of $\tilde{\mathcal{G}}$, this would mean that there is a directed loop in $\tilde{\mathcal{G}}$. This means that each label is used at least twice in $\tilde{\mathcal{G}}$, or in other words, that $\tilde{\mathcal{G}}$ has at least $2n$ vertices. It is well-known that the minimum number of edges that can connect $2n$ vertices is $2n - 1$. Therefore, $\tilde{\mathcal{G}}$ has at least $2n - 1$ edges. By Lemma 6, however, $\mathcal{G}$ has the same number of edges as $\tilde{\mathcal{G}}$, and each edge in $\mathcal{G}$ represents a message in the protocol. $\Box$

4 Two lessons learned

Before presenting our protocols, we recall two common flaws in entity authentication protocols by reviewing two protocols that exhibit these flaws. The first one is a unilateral entity authentication protocol, which is similar to the protocol of Example 2, but this time symmetric key cryptography is used:

$$\begin{array}{lll} 1. & A \to B: & r_a \\ 2. & B \to A: & \{r_a\}_{K_{ab}} \end{array}$$

The protocol works as follows: A sends an unpredictable random number r_a to B. B encrypts the received challenge with the symmetric key K_{ab} that it shares with A, and sends the encrypted random number $\{r_a\}_{K_{ab}}$ back to A. A decrypts the response with the same key, and verifies that the resulted cleartext is indeed its random number r_a. The claim is that if this verification is successful, then A authenticated B.

This is wrong, because A cannot be sure that it was B who encrypted r_a with K_{ab}, since B is not the only one who can encrypt with this key. Ironically, it may be A itself who generated $\{r_a\}_{K_{ab}}$ in a concurrent run of the same protocol initiated by the attacker. The attack scenario that exploits this flaw is illustrated in Figure 4.

In this attack, the attacker impersonates B. In order to do so, it has to respond to the challenge of A by encrypting r_a with K_{ab}. Since it does not possess this key, it cannot itself perform the encryption. Instead, it starts a new instance of the protocol with A pretending to be B, and challenges A with r_a. Recall that, according to our system model introduced in Section 2, A may run several instances of the protocol concurrently, and it may play

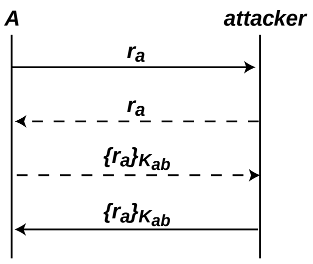

Figure 4: Reflection attack

different roles in different instances. Here, A runs two instances of the protocol, and in the first one it plays the initiator, while in the second one it plays the responder role. Hence, A encrypts the false challenge with K_{ab} and sends the result to the attacker in the second instance. The attacker can now replay it back to A and complete the attack in the first instance.

The usual solution proposed in the literature for this problem is to include a direction label explicitly in each encrypted message. A protocol can, for instance, adopt the convention that each encrypted message contains the name of the principal who generated it (i.e., a from field). In a more economical solution, the direction label can even be a single bit. One can imagine, for instance, that the names of the principals can be lexicographically ordered (bit strings typically have this property). Then each encrypted message sent by A to B, where $A < B$, could contain a 0, while encrypted messages in the reverse direction could contain a 1. When a principal decrypts a message, it looks at the direction label, and if this indicates that the message was generated by the principal itself, then the message is discarded.

The conclusion is the following:

Lesson 1: If symmetric key encryption is used, then some mechanism is needed to ensure that the intended direction of each encrypted message can unambiguously be determined by those who can decrypt the message. □

The next example for a flawed entity authentication protocol is the Woo-Lam protocol [12]:

$$\begin{array}{lll}
1. & A \to B: & A \\
2. & B \to A: & r_b \\
3. & A \to B: & \{r_b\}_{K_{as}} \\
4. & B \to S: & \{A,\ \{r_b\}_{K_{as}}\}_{K_{bs}}
\end{array}$$

5. $S \to B: \quad \{r_b\}_{K_{bs}}$

A major difference between this protocol and the previous one is that this one uses a designated principal called the authentication server S. Instead of sharing keys with each other, principals share a secret key with the authentication server. It is also assumed that the authentication server is trusted for correctly translating a message encrypted with the key of a principal to a message encrypted with the key of another principal.

The Woo-Lam protocol works as follows: A claims that its identity is A. In order to verify this, B challenges A with an unpredictable random number r_b. A proves its identity by encrypting the challenge with the key K_{as}, which it shares with the authentication server S. The response $\{r_b\}_{K_{as}}$ is sent to B. Since B does not possess K_{as}, it cannot verify the response. Therefore, it calls for the help of the authentication server: B sends the message $\{A,\ \{r_b\}_{K_{as}}\}_{K_{bs}}$ to S. S decrypts the request and then decrypts A's response inside; it knows that it has to use K_{as} for decrypting the response, because the request contains the name of A. Then, S encrypts the resulted random number with the key K_{bs} and sends $\{r_b\}_{K_{bs}}$ to B. Finally, B decrypts the message of S and verifies that it received back its random number r_b. The claim is that if this verification is successful, then B authenticated A.

This time, it is not so obvious why this is wrong. Nevertheless, the protocol is known to be vulnerable [1] to the following attack (Figure 5): Let us assume that the attacker compromised the key of a legitimate principal M of the system. This means that the attacker knows the key K_{ms} shared by M and the server S. Using this key, it can impersonate A (who is not compromised) to B. The attacker starts two instances of the protocol with B concurrently; the first instance is started in the name of A and the second one is in the name of M. B generates two random numbers r_b and r'_b and sends them as challenges to A and M, respectively. These messages are intercepted by the attacker and they never arrive to A and M. The attacker then encrypts r_b, which was intended for A, with the key K_{ms} and sends the result $\{r_b\}_{K_{ms}}$ to B in both instances of the protocol. It is very likely that the protocol is implemented in such a way that B does not check responses received in different instances of the protocol for equality. Therefore, B believes that it received the responses from A and M, and sends the corresponding requests $\{A,\ \{r_b\}_{K_{ms}}\}_{K_{bs}}$ and $\{M,\ \{r_b\}_{K_{ms}}\}_{K_{bs}}$, respectively, to S. S decrypts the received responses with K_{as} and K_{ms}, respectively, thus, using the wrong key for the first response. Let us denote the result of decrypting $\{r_b\}_{K_{ms}}$ with K_{as} by x. Because of the properties of symmetric

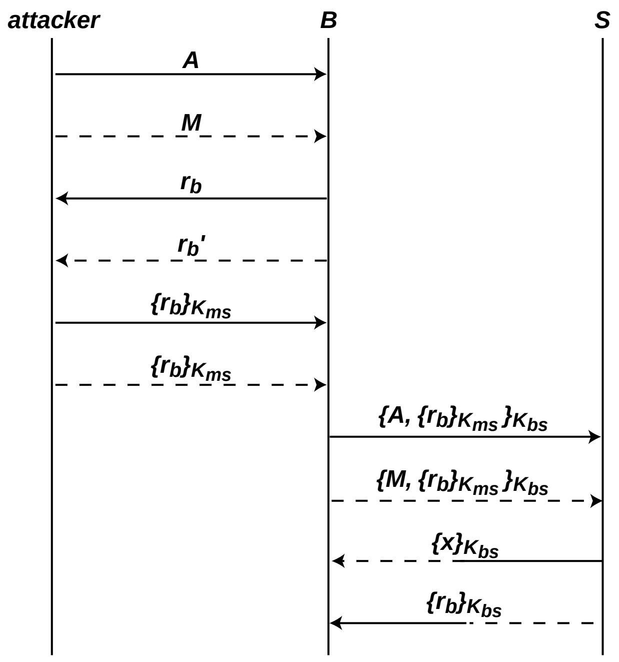

Figure 5: Attack against the Woo-Lam authentication protocol

key ciphers, x looks like a random number. Since the authentication server expects a random number as a result of the decryption, and it cannot check that it is the right number, because it does not know what was the challenge sent by B to A, it accepts x, and does not detect the attack. It responds to the requests of B by sending $\{x\}_{K_{bs}}$ and $\{r_b\}_{K_{bs}}$ to B. When B verifies these messages, it recognizes that the first response is wrong. It does, however, accept the second one, which contains r_b, and since this number was the challenge for A, B attributes the second response to A. Finally, B concludes that A was alive and responded to its challenge, while someone might try to impersonate M.

The source of the flaw is that the authentication server suppresses some critical information when it responds to a request: it does not tell the requesting principal which key it used to decrypt the response. At first glance, one might think that the requesting principal can infer this information from the context, but, as the previous attack shows, this is false. Therefore, it is more secure to mention which key was used by putting a key identifier or the name of the corresponding principal in the last message.

The lesson we can learn from this example is the following:

Lesson 2: If a trusted mediator is used to translate a message encrypted with a given key K to a message encrypted with another key K', then all the semantical information of the original message must be retained. In particular, the translated message should contain the key identifier of K or other equivalent data from which this information can be securely inferred by the destination of the translated message. □

5 Multi-party entity authentication protocols

In Section 3, we proved that the lower bound on the number of messages of n-party challenge-response protocols for entity authentication is $2n-1$. In this section, we present two protocols that achieve this lower bound. Both protocols have the same message passing structure, but they differ in the assumptions about trust among the protocol participants, and thus, in the content (semantics) of messages.

5.1 Message passing structure

Before going into the details of our protocols, it is worth to tell some words about their message passing structure. We recall Lemma 3, which states that if A authenticates B in a given challenge-response protocol, then there must be three edges e, e', and f in the protocol graph such that e is an outgoing edge from u, e' is an incoming edge to u, f is an outgoing edge from v, and $e \prec f \preceq e'$, where u and v are the vertices that correspond to A and B, respectively. This actually means, that in the unfolded protocol graph, there is a directed path, which starts from and ends in a vertex that is labeled with A, and goes through a vertex that is labeled with B. If A authenticates every other party $B, C, \ldots$ in the protocol, then each of these parties, or more precisely vertices that are labeled with their names, must be traversed by a directed path starting from and ending in a vertex that is labeled with A. Note that one single path can do the job (see Figure 6 (a)). If $B, C, \ldots$ also authenticate every other party in the protocol, then there is a similar path for $B, C, \ldots$ as well. We obtain the protocol with the least number of messages by maximally overlapping these paths (see Figure 6 (b)). The resulting protocol graph has exactly $2n-1$ edges (Figure 6 (c)).

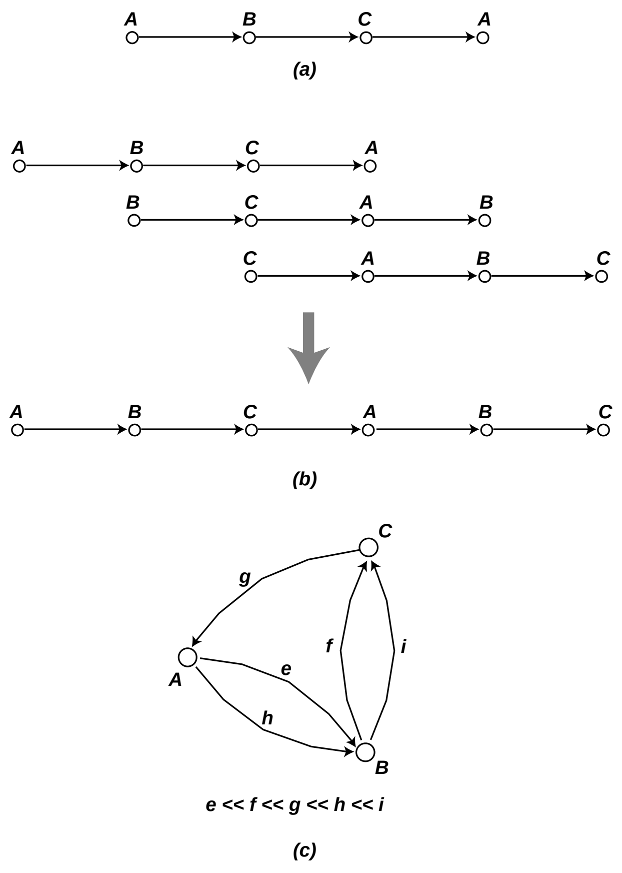

Figure 6: Message passing structure of the basic protocols

5.2 Protocols

In order to make the presentation easier, we describe the three-party versions of our protocols in detail and sketch the general n-party versions only briefly. In addition, we should also mention that our protocols use symmetric key cryptography, but it is straightforward to obtain the versions that use asymmetric key cryptography by replacing symmetric key encryption with digital signatures of the appropriate parties.

Protocol 1:

- *Principle:* The basic idea of Protocol 1 is the following: Each participant generates an unpredictable random number, which is used as a challenge. Challenges are passed around among the protocol participants in a circulating message. Each participant that receives the message and sees the challenges that the message contains includes its identifier in the message before passing it further to the next participant. When a challenge gets back to the principal that generated it, the message contains the list of those principals that saw the challenge and forwarded the message. These forwarding principals must have been alive during the protocol run.

- *Assumptions:* We assume that each pair of principals in the system share a long-term secret key. The secret key shared between A and B, for instance, is denoted by K_{ab}. We also assume that principals trust each other for executing the protocol honestly. In particular, each principal must be trusted for correctly attributing a received message to its sender and faithfully copying all the relevant fields of the received message into the message that is passed further. We will return to this issue of trust later when we analyze Protocol 1 in Subsection 5.3. Finally, we assume that each protocol participant knows (or at least has an assumption about) who the other participants are from the context or additional plaintext fields not mentioned in the description below.

- *Messages of the three-party version:*

$$\begin{array}{lll}
1. & A \rightarrow B: & r_a \\
2. & B \rightarrow C: & r_b,\ \{B,\ r_a\}_{K_{bc}} \\
3. & C \rightarrow A: & r_c,\ \{C,\ r_b,\ B,\ r_a\}_{K_{ac}} \\
4. & A \rightarrow B: & \{A,\ r_c,\ C,\ r_b\}_{K_{ab}} \\
5. & B \rightarrow C: & \{B,\ A,\ r_c\}_{K_{bc}}
\end{array}$$

- *Description of the three-party version:* A generates an unpredictable random number r_a, and sends it to B in message 1. Upon reception of message 1, B generates an unpredictable random number r_b, encrypts its own identifier B and the random number r_a with the key K_{bc}, and sends r_b and the result of the encryption to C in message 2. The identifier in the encrypted part serves as an explicit direction label that allows B to recognize its own messages. Upon reception of message 2, C decrypts the encrypted part, and verifies that it was indeed generated by B by checking the identifier in the first field. If this verification is successful, then C generates an unpredictable random number r_c, encrypts its own identifier C, the random number r_b, the identifier of B, and the random number r_a with the key K_{ac}, and sends r_c and the result of the encryption to A in message 3. The identifier of C serves again as a direction label. When A receives message 3, it decrypts the encrypted part of it, and verifies that it was indeed generated by C by checking the identifier in the first field. Furthermore, it checks if it received back its random number r_a and if the message contains the identifier of B too. If these verifications are successful, then A authenticated B and C, and it continues by encrypting its own identifier A, the random number r_c, the identifier of C, and the random number r_b with the key K_{ab}. A sends the result of the encryption to B in message 4. When B receives message 4, it decrypts it, and verifies that it was indeed generated by A by checking the identifier in the first field. Furthermore, it checks if it received back its random number r_b and if the message contains the identifier of C too. If these verifications are successful, then B authenticated A and C, and it continues by encrypting its own identifier B, the identifier of A, and the random number r_c with the key K_{bc}. B sends the result of the encryption to C in message 5. Finally, when C receives message 5, it decrypts it, and verifies that it was indeed generated by B by checking the identifier in the first field. It also checks if it received back its random number r_c and if the message contains the identifier of A too. If these verifications are successful, then C authenticated A and B and the protocol terminates.

- *Messages of the n-party version:*

1.	$P_1 \to P_2$:	r_1
2.	$P_2 \to P_3$:	$r_2,\ \{P_2,\ r_1\}_{K_{2,3}}$
3.	$P_3 \to P_4$:	$r_3,\ \{P_3,\ r_2,\ P_2,\ r_1\}_{K_{3,4}}$
4.	$P_4 \to P_5$:	$r_4,\ \{P_4,\ r_3,\ P_3,\ r_2,\ P_2,\ r_1\}_{K_{4,5}}$
...	...	...
n.	$P_n \to P_1$:	$r_n,\ \{P_n,\ r_{n-1},\ P_{n-1},\ r_{n-2},\ P_{n-2}, \ldots,\ r_2,\ P_2,\ r_1\}_{K_{1,n}}$
$n+1$.	$P_1 \to P_2$:	$\{P_1,\ r_n,\ P_n,\ r_{n-1},\ P_{n-1}, \ldots,\ r_3,\ P_3,\ r_2\}_{K_{1,2}}$
$n+2$.	$P_2 \to P_3$:	$\{P_2,\ P_1,\ r_n,\ P_n,\ r_{n-1},\ P_{n-1}, \ldots,\ r_4,\ P_4,\ r_3\}_{K_{2,3}}$
...	...	...
$2n-1$.	$P_{n-1} \to P_n$:	$\{P_{n-1},\ P_{n-2},\ P_{n-3}, \ldots,\ P_1,\ r_n\}_{K_{n-1,n}}$

- *Remark for the n-party version:* Let us consider any of the encrypted messages of the protocol above. For a given random number r in this message, the identifiers that stand before r correspond to those parties who have already seen and forwarded r. For instance, in message n, the identifiers before r_2 are $P_3,\ P_4, \ldots,\ P_n$, and indeed, apart from P_1, all the participants have already seen r_2 when message n is sent. Therefore, when a party receives back its random number in a message, it must check if all the other parties are listed before its random number in the message.

Note that Protocol 1 takes into account Lesson 1 and Lesson 2 of Section 4. First, we used sender identifiers (from fields) as explicit direction labels in messages in order to prevent reflection attacks. Second, since each party acts as a trusted mediator, and translates messages encrypted with one key for messages encrypted with another key, we ensured that all the semantical information of the original message is retained by keeping all fields that are relevant for the further processing of the translated message (including the identifier of the sender of the original message).

Protocol 2:

- *Principle:* The main drawback of Protocol 1 is that it relies on the assumption that the protocol participants trust each other for honestly executing the protocol. In Protocol 2, we remove this assumption. The main idea of Protocol 2 is that we allow each protocol participant to directly verify who responded its challenge. Like in Protocol 1, the challenge of each participant is passed around among the other participants, but unlike in Protocol 1, this time it is encrypted with the key that is shared by the challenging and the responding principals before it is passed further to the next participant. Indeed, responding parties do not encrypt the challenge itself, but the encrypted challenge that they receive from the previous responding party. The challenging party finally receives back its random number encrypted by every other party, one after the other. The challenging party verifies the response by decrypting it with the keys it shares with the other parties. If, after performing all the decryptions, it recovers its original random number, then it is convinced that *all* the other parties were alive during the protocol run.

- *Assumptions:* We assume that each pair of principals in the system share a long-term secret key, and each protocol participant knows (or at least has an assumption about) who the other participants are from the context or additional plaintext fields not mentioned in the description below.

- *Messages of the three-party version:*

$$\begin{array}{lll}
1. & A \to B: & r_a \\
2. & B \to C: & r_b,\ \{B,\ r_a\}_{K_{ab}} \\
3. & C \to A: & r_c,\ \{C,\ r_b\}_{K_{bc}},\ \{C,\ \{B,\ r_a\}_{K_{ab}}\}_{K_{ac}} \\
4. & A \to B: & \{A,\ r_c\}_{K_{ac}},\ \{A,\ \{C,\ r_b\}_{K_{bc}}\}_{K_{ab}} \\
5. & B \to C: & \{B,\ \{A,\ r_c\}_{K_{ac}}\}_{K_{bc}}
\end{array}$$

- *Description of the three-party version:* A generates an unpredictable random number r_a, and sends it to B in message 1. Upon reception of message 1, B generates an unpredictable random number r_b, encrypts its own identifier B and the random number r_a with the key K_{ab}, and sends r_b and the result of the encryption to C in message 2. Like in Protocol 1, in Protocol 2 as well, the identifier of the encrypting party in an encrypted message always serves as an explicit direction label to foil reflection attacks. Upon reception of message 2, C generates an unpredictable random number r_c, encrypts its own identifier C and the random number r_b with K_{bc}, encrypts its own identifier C and the encrypted part of message 2 with K_{ac}, and sends r_c and the results of the encryptions to A in message 3. When A receives message 3, it first verifies the last encrypted part of it by decrypting it with the keys K_{ac} and K_{ab}, and checking the identifiers and the random number found inside. If the identifiers match those of C and B, and the random number matches r_a, then A authenticated B and C, and it continues by encrypting its own identifier A and the random number r_c with the key K_{ac}, and encrypting its own identifier A and the other encrypted part of message 3 with the key K_{ab}. Then A sends the results of the encryptions to B in message 4. When B receives message 4, it verifies the last encrypted part of it by decrypting it with the keys K_{ab} and K_{bc}, and checking the identifiers and the random number found inside. If the identifiers match those of A and C, and the random number matches r_b, then B authenticated A and C, and it continues by encrypting its own identifier B and the other encrypted part of message 4 with the key K_{bc}. Then B sends the result of the encryption to C in message 5. Finally, when C receives message 5, it verifies it by decrypting it with the keys K_{bc} and K_{ac}, and checking the identifiers and the random number found inside. If the identifiers match those of

B and A, and the random number matches r_c, then C authenticated A and B, and the protocol terminates.

- *Messages of the n-party version:*

1.	$P_1 \to P_2$:	r_1
2.	$P_2 \to P_3$:	$r_2,\ \{P_2,\ r_1\}_{K_{1,2}}$
3.	$P_3 \to P_4$:	$r_3,\ \{P_3,\ r_2\}_{K_{2,3}},\ \{P_3,\ \{P_2,\ r_1\}_{K_{1,2}}\}_{K_{1,3}}$
...	...	...
n.	$P_n \to P_1$:	$r_n,\ \{P_n,\ r_{n-1}\}_{K_{n-1,n}},$
		$\{P_n,\ \{P_{n-1},\ r_{n-2}\}_{K_{n-2,n-1}}\}_{K_{n-2,n}}, \ldots,$
		$\{P_n,\ \{P_{n-1}, \ldots \{P_2,\ r_1\}_{K_{1,2}} \ldots\}_{K_{1,n-1}}\}_{K_{1,n}}$
$n+1$.	$P_1 \to P_2$:	$\{P_1,\ r_n\}_{K_{1,n}},$
		$\{P_1,\ \{P_n,\ r_{n-1}\}_{K_{n-1,n}}\}_{K_{1,n-1}}, \ldots,$
		$\{P_1,\ \{P_n, \ldots \{P_3,\ r_2\}_{K_{2,3}} \ldots\}_{K_{2,n}}\}_{K_{1,2}}$
$n+2$.	$P_2 \to P_3$:	$\{P_2,\ \{P_1,\ r_n\}_{K_{1,n}}\}_{K_{2,n}},$
		$\{P_2,\ \{P_1,\ \{P_n,\ r_{n-1}\}_{K_{n-1,n}}\}_{K_{1,n-1}}\}_{K_{2,n-1}}, \ldots,$
		$\{P_2,\ \{P_1, \ldots \{P_4,\ r_3\}_{K_{3,4}} \ldots\}_{K_{1,3}}\}_{K_{2,3}}$
...	...	...
$2n-1$.	$P_{n-1} \to P_n$:	$\{P_{n-1},\ \{P_{n-2}, \ldots \{P_1,\ r_n\}_{K_{1,n}} \ldots\}_{K_{n-2,n}}\}_{K_{n-1,n}}$

Note that Protocol 2 takes into account Lesson 1 by using the identifiers of the encrypting principal in each encrypted message as an explicit direction label that prevents reflection attacks. It does not, however, use Lesson 2, because here principals do not translate messages encrypted with one key for messages encrypted with another key.

5.3 Analysis and comparison

In order to better understand Protocol 1 and Protocol 2, and the differences between them, we sketch how their main assumptions and achievements could be formalized in a formal logic called SvO [11]. We analyze the three-party protocols from the point of view of party A, but the same reasoning applies for n-party protocols ($n > 3$) and the other parties as well.

The first question is: How can the goal of entity authentication be modeled in the SvO logic? We recall that entity authentication means that a party, say A, is convinced that another party, say B, was alive and sent some messages in a bounded time interval in the local time of A. In the SvO logic, the following formula can represent this:

$$A \textit{ believes } (B \textit{ says } X)$$

where X is some message or a part of a message (typically some function of a fresh nonce generated by A).

Now, let us investigate how such a formula can be derived for Protocol 1. The last message that A receives in Protocol 1 is message 3. Only the encrypted part is interesting for A, which we idealize as:

$$\{C,\ r_b,\ B,\ r_a,\ (B\ \mathit{said}\ r_a)\}_{K_{ac}}$$

Using the assumption that A believes that K_{ac} is a good shared secret key between A and B, we can easily derive that

$$A\ \mathit{believes}\ (C\ \mathit{said}\ (C,\ r_b,\ B,\ r_a,\ (B\ \mathit{said}\ r_a)))$$

Since A believes that its own random number r_a is fresh, A believes that the message received from C is fresh. This allows us to derive that

$$A\ \mathit{believes}\ (C\ \mathit{says}\ (C,\ r_b,\ B,\ r_a,\ (B\ \mathit{said}\ r_a)))$$

from which we can easily get that

$$A\ \mathit{believes}\ (C\ \mathit{says}\ r_a) \tag{1}$$

and

$$A\ \mathit{believes}\ (C\ \mathit{says}\ (B\ \mathit{said}\ r_a)) \tag{2}$$

(1) means that A authenticated C. In order to go further and derive that A authenticated B as well, the following must hold:

$$A\ \mathit{believes}\ (C\ \mathit{controls}\ (B\ \mathit{said}\ r_a)) \tag{3}$$

Then, from (2) and (3) we can derive that

$$A\ \mathit{believes}\ (B\ \mathit{said}\ r_a)$$

Since we assumed that A believes that its own random number r_a is fresh, we get that

$$A\ \mathit{believes}\ (B\ \mathit{says}\ r_a)$$

This means that A authenticated B.

Note, however, that (3) cannot be derived from the protocol, thus, we must make the assumption that it holds. Indeed, (3) models A's trust in

C for correctly attributing messages to B and for correctly forwarding the content of these messages to A.

Now, we turn our attention to Protocol 2. The last message received by A is again message 3. Only the last encrypted part is interesting for A, which we idealize in the following way:

$$\{C,\ \{B,\ r_a\}_{K_{ab}}\}_{K_{ac}}$$

Assuming that A believes that K_{ab} and K_{ac} are good shared secrets between A and B, and between A and C, respectively, we can derive that

$$A\ believes\ (C\ said\ \{B,\ r_a\}_{K_{ab}}) \tag{4}$$

and

$$A\ believes\ (B\ said\ r_a) \tag{5}$$

Since A believes that its own random number r_a is fresh, we can easily derive in the SvO logic that A believes that $\{B,\ r_a\}_{K_{ab}}$ is fresh as well. Thus, from (4) we get that

$$A\ believes\ (C\ says\ \{B,\ r_a\}_{K_{ab}})$$

and from (5) we get that

$$A\ believes\ (B\ says\ r_a)$$

This means that A authenticated both B and C.

Note that, although Protocol 1 and Protocol 2 achieve the same goal, unlike Protocol 1, Protocol 2 does not need any assumptions about existing trust between the parties.

6 Conclusion

In this paper, we addressed the problem of multi-party entity authentication. We proved that the lower bound on the number of messages of multi-party challenge-response protocols for entity authentication is $2n - 1$, where n is the number of the parties participating in the protocol. Our proof is based on modeling the protocol with a directed graph, the vertices of which correspond to the parties, and the edges of which correspond to the messages of the protocol, and defining a partial ordering on the edges according to the timing of messages in the protocol. Besides allowing us to prove the lower bound on the number of messages, this model proved to be helpful in understanding some interesting structural properties of challenge-response type entity authentication protocols.

We presented two protocols that are efficient in the sense that they use the minimum number of messages required to solve the multi-party entity authentication problem based on challenge-response principles (i.e., they achieve the lower bound on the number of messages). The protocols have the same message passing structure, but they differ in the assumptions about trust among the parties, and thus, in the content (semantics) of the messages. We analyzed and compared our protocols with the help of the SvO authentication protocol logic. We note that our protocols can easily be extended to server assisted entity authentication and session key establishment, which we did not discuss in this paper due to space limitations.

Finally, we should note that, in this paper, we were mainly concerned with minimizing the number of messages in challenge-response protocols for entity authentication. On the one hand, this makes sense, because each message sent involves the use of several lower level protocols down in the communication protocol stack, and thus, produces some overhead. By minimizing the number of messages of the authentication protocol, we can minimize this overhead. On the other hand, minimizing the number of messages does not necessarily minimizes the required bandwidth. If the protocol uses few messages, but these are long, then we do not gain much in bandwidth, and a protocol with more but smaller messages might be more desirable. Thus, in general, we are facing a more complex optimization problem, in which both the number of messages and the total amount of data exchanged in the protocol must be taken into consideration. We leave this issue for future study.

References

[1] M. Abadi and R. Needham. Prudent engineering practice for cryptographic protocols. In *Proceedings of the IEEE CS Symposium on Research in Security and Privacy*, pages 122–136, 1994.

[2] R. Anderson and R. Needham. Robustness principles for public key protocols. In *Advances in Cryptology – CRYPTO'95*, pages 236–247, 1995.

[3] R. Bird, I. Gopal, A. Herzberg, P. Janson, S. Kutten, R. Molva, and M. Yung. Systematic design of a family of attack-resistant authentication protocols. *IEEE Journal on Selected Areas in Communications*, 11(5):679-693, 1992.

[4] M. Burrows, M. Abadi, and R. Needham. A logic of authentication. *ACM Transactions on Computer Systems*, 8(1):18–36, February 1990.

[5] J. Clark and J. Jacob. A survey of authentication protocol literature. `http://www-users.cs.york.ac.uk/˜ jac/papers/drareview.ps.gz`

[6] L. Gong, R. Needham, and R. Yahalom. Reasoning about belief in cryptographic protocols. In *Proceedings of the IEEE CS Symposium on Research in Security and Privacy*, pages 234–248, 1990.

[7] A. Menezes, P. van Oorschot, and S. Vanstone. *Handbook of Applied Cryptography.* CRC Press, 1997.

[8] J. Millen, S. Clark, and S. Freedman. The Interrogator: Protocol security analysis. *IEEE Transactions on Software Engineering*, SE-13(2):274–288, February 1987.

[9] C. Mitchell. Limitations of challenge-response entity authentication. *IEE Electronics Letters*, 25(17), 1989.

[10] A. Mitropoulos and H. Meijer. Zero knowledge proofs – a survey. Technical Report No. 90-IR-05, Queen's University at Kingston, Kingston, Ontario, Canada, 1990.

[11] P. Syverson and P. van Oorschot. On unifying some cryptographic protocol logics. In *Proceedings of the IEEE CS Symposium on Research in Security and Privacy*, pages 14–28, 1994.

[12] T. Woo and S. Lam. Authentication for distributed systems. *Computer*, 25(1):39–52, January 1992.