A Family of Multi-Party Authentication Protocols

Extended abstract

C.J.F. Cremers and S.Mauw

Eindhoven University of Technology,
Department of Mathematics and Computer Science,
P.O. Box 513, NL-5600 MB Eindhoven, The Netherlands
{ccremers,sjouke}@win.tue.nl

Abstract. We introduce a family of multi-party authentication protocols and discuss six novel protocols, which are members of this family. The first three generalize the well-known Needham-Schroeder-Lowe public-key protocol, the Needham-Schroeder private-key protocol, and the Bilateral Key Exchange protocol. The protocols satisfy *injective synchronisation*, which is a strong authentication property, and establish agreement over the nonces. These protocols make use of *delegated authentication* to keep the protocols small and efficient. For each of these protocols we define a strengthened version that does not rely on delegated authentication. All instantiations of the protocol family consist of $2p-1$ messages for p parties, which we show to be the minimal number of messages required to achieve the desired security properties in the presence of a Dolev-Yao style intruder with compromised agents.

1 Introduction

In the context of Dolev-Yao style modeling of security protocols, several protocols have been proposed in order to satisfy forms of *mutual authentication* (for an overview of authentication protocols see [14, 39]). Of these, the best known is Needham-Schroeder-Lowe (NSL-public key protocol, or NSL for short) from [31, 35]. The NSL protocol satisfies even the strongest forms of authentication [20], and has been studied extensively.

The operation of the three-message base protocol is as follows (see Figure 1). In the first step, the initiator of the protocol, executing the a role, creates a random value (often called a nonce or a challenge) na. He encrypts this value along with his name with the public key pkb of the intended responder. When the responder, executing the b role, receives such a message, he generates his own random value nb. He responds to the challenge by encrypting both nonces as well as his own name, with the public key of the initiator. In the third step, the initiator sends back the random value nb to the responder, encrypted by the public key of the responder.

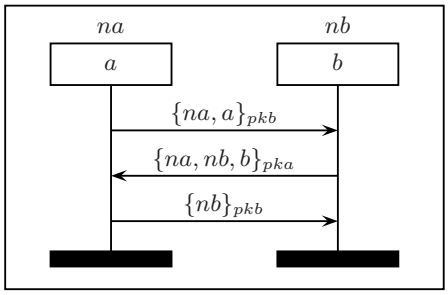

Fig. 1. The Needham-Schroeder-Lowe protocol with public keys.

Similar protocols, such as the Needham-Schroeder (NS for short) private-key protocol and the Bilateral Key Exchange protocol, have the same underlying structure as the NSL protocol. The under-

lying communication pattern consists of two challenge-response steps, which can be realized through three messages.

These protocols were designed for two parties who want to authenticate each other, which is often referred to as bilateral authentication. In many settings such as modern e-commerce protocols there are three or more parties that need to authenticate each other. In such a setting we could naively instantiate multiple bilateral authentication protocols to mutually authenticate all partners. For p parties, such mutual authentication would require $(p \times (p-1))/2$ instantiations of the protocol, and three times as many messages. In practice, when multi-party authentication protocols are needed, protocol designers instead opt to design new protocols (often 3 or 4-way handshakes, see e.g. [25]), which possibly introduces new faults. Central to our research is the question of how to generalize these NSL-like protocols to a multi-party setting.

In this paper we present six novel multi-party authentication protocols, which are all instances of the same family. Three of these are based on *delegation of authentication*, which means that any of the parties trusts any of the other parties to do his job in authenticating the other parties. We also define three strengthened versions of these protocols that do not rely on the delegation of authentication.

All protocols have the same underlying communication structure and will only differ in the contents of the messages. For p parties, the underlying communication structure consists of $2p-1$ messages, which turns out to be the optimal message complexity.

Every member of the protocol family is itself parameterized with a value p, representing the number of participants. We do not only require that the protocol is correct for each p in isolation, we require that the protocol is correct even when run in parallel with several instances of the same protocol for different values of p.

We adopt the standard Dolev-Yao model with compromised agents to model the capabilities of the adversary. This means that the intruder has complete control over the communication network and that he can unpack encrypted messages if he knows the decryption key. We assume that a number of agents can conspire with the intruder to break security of a protocol.

The security protocols will have to satisfy a strong form of authentication, called *injective synchronisation* [22]. Since the development of correct security protocols has proven to be a notoriously difficult task we have use the framework introduced in [18, 20–22] to prove the proposed protocols correct. Due to space limitations the proofs are not included in this paper. For a correctness proof of the generalized NSL protocol, we refer to our technical report [19].

We proceed as follows. In Section 2 we define the communication structure underlying our protocols. Six instantiations are presented in Section 3. Related work is discussed in Section 4 and we draw conclusions and discuss further work in Section 5.

2 A framework for multi-party authentication protocols

To give a feel of the structure of the kind of authentication protocols that we study, we first look at the structure of the NSL protocol. In that protocol, each agent has a challenge-response cycle to validate the other agent's identity. These two challenge-response cycles are linked together by identifying the response of the second agent with its challenge. Its generalization follows the same line of thinking. Every agent conducts a challenge-response cycle with its neighbouring agent, while combining its own challenge with a response to another agent's challenge whenever possible.

The starting point is therefore a collection of p parties $R(0), \ldots, R(p-1)$, each controlling one of the nonces $n(0), \ldots, n(p-1)$, as in Figure 2. The first agent, $R(0)$ sends its challenge to the next agent, $R(1)$, which forwards it to the third agent, and so on. The challenge cycle for the first agent closes if he receives his challenge back from $R(p-1)$, which means that all agents have responded to his

challenge. The second agent has a similar challenge-response cycle, while piggybacking as much as possible on the messages sent on behalf of $R(0)$. This will yield one extra message, namely from $R(0)$ to $R(1)$ at the end of $R(1)$'s cycle. Repeating this for all agents yields the zig-zag structure of Figure 2. The figure also illustrates that we consider two types of messages. The first p messages (of type MsgA) contain both challenges and responses, while the last $p-1$ messages (of type MsgB) contain responses only. This distinction is merely made to provide simpler definitions of the instantiated protocols.

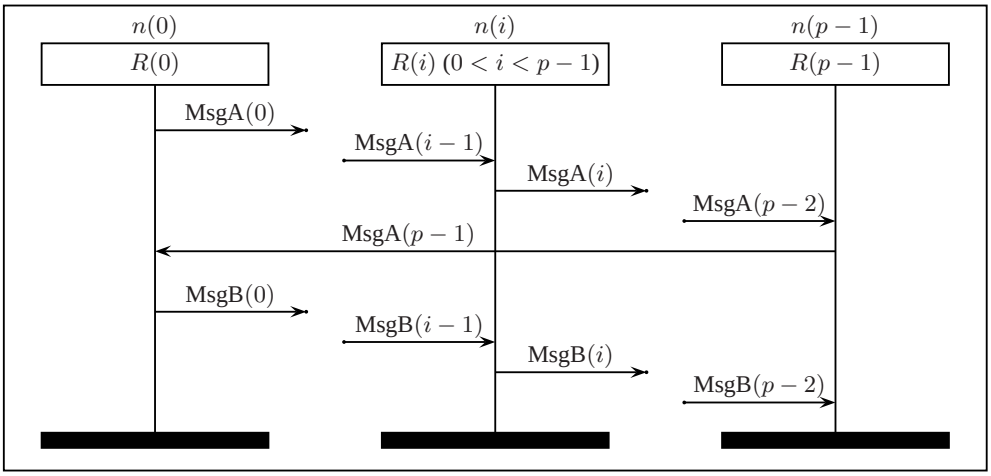

Fig. 2. Multi-party communication structure.

In order to be precise, we define the i'th protocol message, for $0 \leq i < 2p-1$, by

$$\text{Msg}(i) = \begin{cases} \text{MsgA}(i) & \text{if } 0 \leq i < p, \\ \text{MsgB}(i-p) & \text{if } p \leq i < 2p-1. \end{cases}$$

Members of this protocol family can now be constructed by instantiating the messages of type MsgA and MsgB in such a way that correct authentication is guaranteed. In the next section we will provide some of the more interesting examples.

A protocol in this family can be deployed in two main ways. First, it can be instantiated for a specific number of parties, to yield e.g. a four-party authentication protocol. In this way it can be used instead of custom n-way handshake protocols. Second, it can be used in its most generic form, and have the initiating role $R(0)$ choose the number of participants p. Agents receiving messages can deduce the chosen p and their supposed role from the contents of the messages.

Based on this communication pattern, we cannot make any statements on the correctness of its instantiations, since that will depend on the actual message contents. However, we can already discuss the message complexity and the basic authentication requirement of this family of protocols. As stated in the introduction, we require that the protocols satisfy *injective synchronisation*, as defined in [22]. Informally, for a two-party protocol, it states the following:

> Initiator I considers a protocol synchronising, whenever I as initiator completes a run of the protocol with uncompromised responder R, then R as responder has been running the protocol with I. Moreover, all messages are received exactly as they were sent, in the order as described by the protocol. Initiator I considers a protocol injectively synchronising if the protocol synchronises and each run of I corresponds to a unique run of R.

This definition extends in a natural way to multi-party protocols. Synchronisation is based on the notion of *intensional specification*, as introduced by Roscoe [37]. Lowe [32] expresses that the definition of intensional specification is strictly stronger than the more common notion of *agreement*. This is formalized in [18, 20, 22], so the developed protocols satisfy agreement as well. We refer the reader to [20–22] for a detailed definition of injective synchronisation and a proof that agreement is implied by injective synchronisation, as well as comparisons between synchronisation and various other notions of authentication.

As discussed in [22], the *loop-property* is instrumental to achieve injectivity. This property states that for every two agents A and B there must be a message sequence, starting and ending in A, which passes through B (possibly via some other agents). In the current context the loop-property turns out to be a necessity. Phrased in terms of challenge-response behaviour: in order to achieve injective synchronisation, each role must send a challenge that is replied to by all other roles.

Now we can prove that the minimal number of messages needed to achieve injective synchronisation of p parties is $2p - 1$. Consider the first message sent by some role $R(x)$, and call this message m. In order to achieve a loop to all other roles after this first message, every role will have to send at least one message after m. Including message m this will yield at least p messages. Next we observe that every role must take part in the protocol and we consider the first message sent by each of the roles. If we take $R(x)$ to be the last of the p roles that becomes active in the protocol, it must be the case that before $R(x)$ sends his first message, at least $p - 1$ messages have been sent. Adding this to the p messages that must have been sent after that message, yields a lower bound of $2p - 1$ messages.

Important to the approach here is that we consider *local synchronisation claims*. That means that an agent may decide that the complete protocol has been executed exactly as expected, based on his local observations only. These observations only take into account the contents of the communications that the agent was involved in. Whenever such an agent successfully completes a run of a synchronising protocol, all other parties involved in the protocol have executed their part exactly as expected.

3 Instantiations

In this section we will define six multi-party authentication protocols, which are all based on the framework from Section 2. The first protocol (denoted by α) generalizes the NS private-key protocol. Agents use signatures to construct their replies to the received challenges. Rather than signing whole incoming messages, agents only sign the contents of these message, thereby effectively replacing the signature of the previous agent. This expresses that agents can trust all other agents taking part in the same session. We call this strategy *delegated authentication*. Stated differently, if one of the agents taking part in a session is compromised, the adversary can abuse this to falsely authenticate one or more other participants to the same session. The second protocol (denoted by β) additionally provides secrecy of the nonces, by using public keys for encrypting the nonces. It generalizes the well-known NSL public-key protocol. The third protocol (denoted by β^*) is an optimization of the second protocol in the sense that it uses symmetric encryption instead of asymmetric encryption for half of the messages.

These protocols have to be strengthened if it is desired to reach agreement with the uncompromised participants of a session, even if some participants of that session are compromised. Therefore, we define three non-delegated variants of the protocols, denoted by α-nd, β-nd and β-nd^*, that do not rely on delegated authentication. This requires that every agent proves its identity to every other agent, which in general yields protocols that are less efficient.

Please notice the distinction between two interpretations of compromised agents. The first interpretation considers the correctness of a session only if all participants are uncompromised, possibly

executed in parallel with other session in which compromised agents take part. This is the standard interpretation, adopted in most literature on security protocols. In this interpretation delegation of authentication is acceptable. However, in the setting of multi-party authentication protocols the above sketched interpretation can be useful. It states that decent behaviour of the protocol is required even if compromised agents take part in the session under study. This interpretation requires direct authentication.

3.1 Generalized NS private-key protocol (α).

The first instantiation of our framework generalizes the NS private-key protocol. Figure 3 illustrates the four-party version of this protocol.

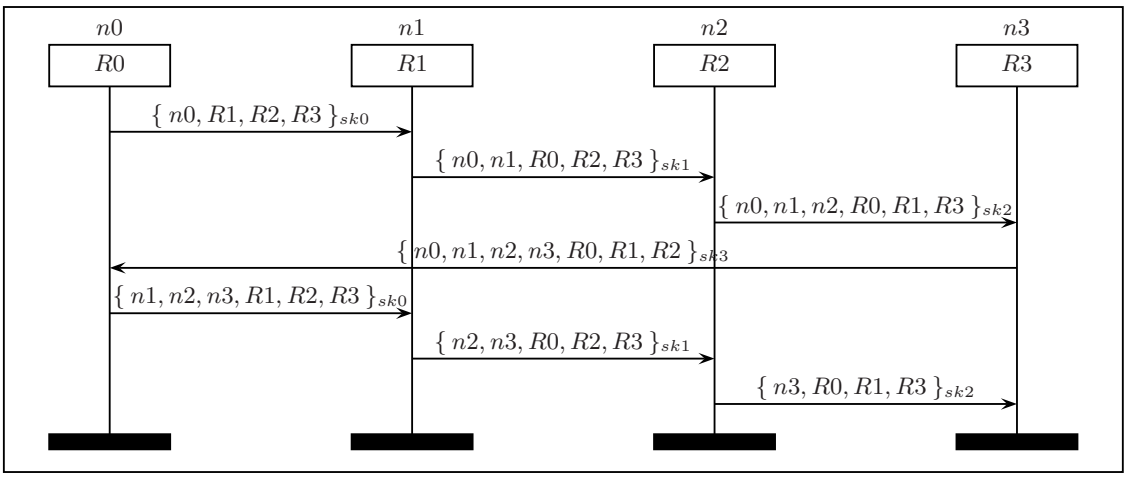

Fig. 3. Four-party generalized NS private-key protocol (α).

First, the initiating agent chooses which parties he wants to communicate with. He creates a new random value, $n0$, and combines this with the names of agents $R1$, $R2$ and $R3$. He signs the resulting message with his private key, and sends the message to $R1$. Upon receipt of this message and verification of the signature, the second agent takes the contents of the message, adds his own fresh nonce and replaces his name $R1$ by $R0$. This modified message is then signed and sent along. This continues until each agent has added his nonce, upon which the message is sent back to the initiating agent. This agent checks whether the message contains the nonce he created earlier, and whether all agent names match. Then he can conclude that the other agents are authenticated. Next, in order to prove his own identity, he sends a signed message containing the other agents' nonces to $R1$. The subsequent agents again check whether their own nonces are in the message, remove this nonce, and pass the resulting message on.

For the general definition of this protocol we have to make the two message types explicit, which requires the auxiliary function $\mathrm{AL}(x)$, defining the list of all parties exclusive of x.

$$\begin{aligned} \mathrm{AL}(x) &= [R0, R1, \ldots, R(p-1)] \setminus x \\ \mathrm{MsgA}_\alpha(i) &= \{\, [n0, \ldots, ni], \mathrm{AL}(Ri) \,\}_{sk(Ri)} \\ \mathrm{MsgB}_\alpha(i) &= \{\, [n(i+1), \ldots, n(p-1)], \mathrm{AL}(Ri) \,\}_{sk(Ri)} \end{aligned}$$

This protocol achieves injective synchronisation through delegated authentication.

3.2 Generalized NSL public-key protocol (β).

Because the previous protocol makes use of a signature scheme, the nonces will be exposed to an eavesdropping adversary. This is not problematic when using these nonces to generate a unique session id, but the nonces cannot be used for establishing a secret session key. Secrecy of nonces can be achieved by using the public key of the sender for encryption, as in the NSL public-key protocol. The four-party generalization of this protocol is in Figure 4.

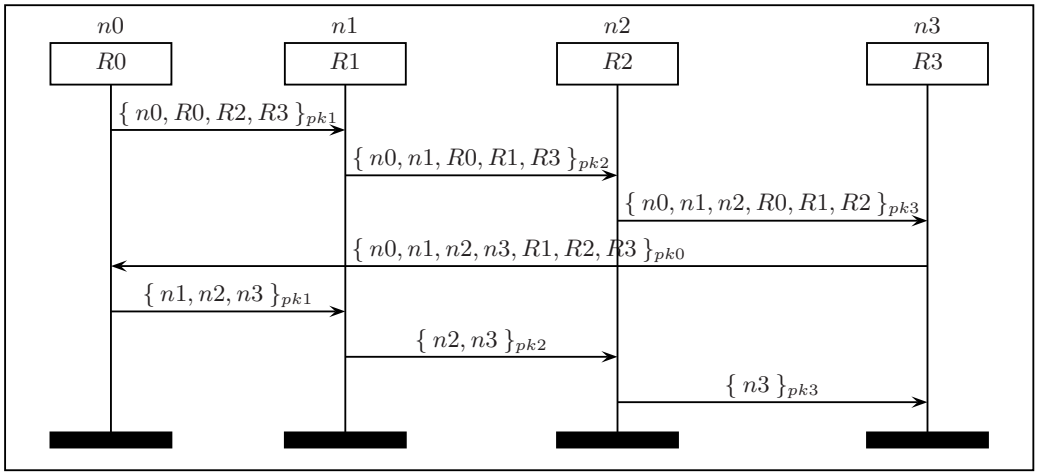

Fig. 4. Four-party generalized NSL public-key protocol (β).

The messages of the generalized protocol are defined using the function $next$, which determines the next role in the list of participants in a cyclic way.

$$
\begin{aligned}
next(i) &= R((i+1) \mod p) \\
\text{MsgA}_\beta(i) &= \{\, [n(0) \ldots n(i)], \text{AL}(next(i)) \,\}_{pk(next(i))} \\
\text{MsgB}_\beta(i) &= \{\, [n(i+1) \ldots n(p-1)] \,\}_{pk(next(i))}
\end{aligned}
$$

Clearly, if we instantiate the generalized protocol for $p = 2$, we get exactly the three message version of the NSL protocol. The purpose of this protocol is to achieve authentication of all parties and distribution and secrecy of all nonces.

We refer to our technical report [19] for a correctness proof of this protocol. In this proof we have used some (but not all) information that distinguishes the messages in the protocol. In particular, we have only used the collection of agent names and nonces occurring in the messages. A direct consequence of this is that the exact order of the agent list and nonce list is not relevant, as long as it is used consistently. We could e.g. redefine messages of type A as to start with a reversed list of roles, followed by the list of nonces. Furthermore we did not require in the proof of the protocol, that there was nothing else inside the encrypted terms besides names and nonces. Thus, we can add any payload inside the encryption, as long as we ensure that it cannot be confused with an agent term or a nonce.

This opens up several possibilities for establishing e.g. keys between pairs of agents inside of the generalized NSL protocol.

Finally we want to mention that our correctness proof implies that the protocol is not only correct for any specific choice of p. Rather, the protocol is even correct in a context where the instances of the protocol for different values of p run in parallel.

3.3 Generalized Bilateral Key Exchange (β^*).

Similar to the Bilateral Key Exchange protocol (BKE) as described in [14] we can opt to replace the asymmetric encryption for the messages of type B by symmetric encryption with the nonce of the recipient. We can then omit this nonce from the list. We use ϵ to denote a constant representing the empty list. In general symmetric encryption is much less computationally intensive than asymmetric encryption, resulting in a more efficient protocol than the previous one. Figure 5 illustrates the four-party BKE protocol. The generalized BKE protocol satisfies the same security properties as the generalized NSL public-key protocol.

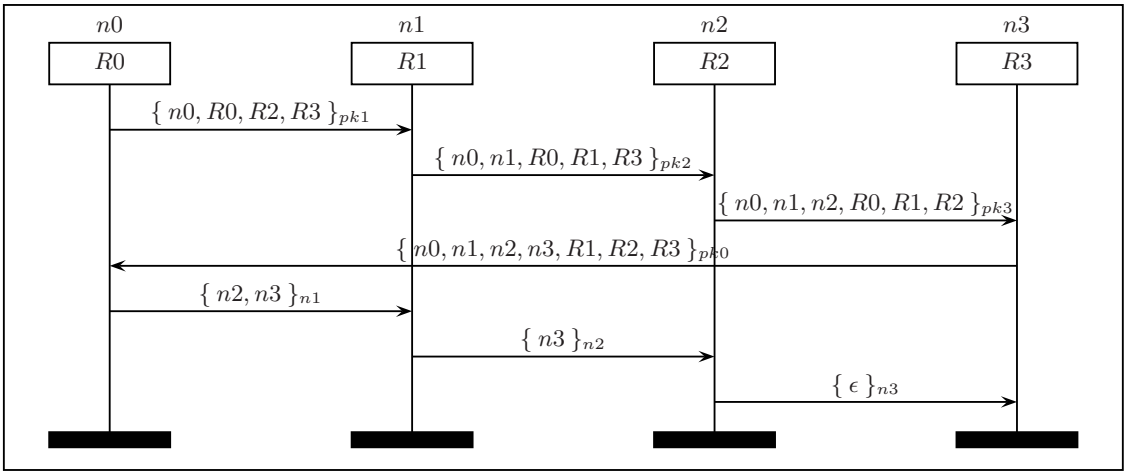

Fig. 5. Four-party generalized BKE protocol (β^*).

The generalized BKE protocol is described by the following message definitions.

$$\text{nlist}(i) = \begin{cases} [n(i+2)\ldots n(p-1)] & \text{if } i < p-1 \\ \epsilon & \text{if } i = p-1 \end{cases}$$
$$\text{MsgA}_{\beta^*}(i) = \{\ [n0\ldots ni], \text{AL}(next(i))\ \}_{pk(next(i))}$$
$$\text{MsgB}_{\beta^*}(i) = \{\ \text{nlist}(i)\ \}_{n(i+1)}$$

3.4 Generalized NS private-key protocol without delegated authentication (α-nd).

As stated before, the above protocols all make use of the delegation of authentication. If one of $R0, \ldots, R(p-1)$ is compromised in a given session, the attacker can falsely authenticate the other parties throughout that session. The solution of this problem is to not rely on the chain of trust, which means that every party has to explicitly verify the responses of all other parties. The simplest way to achieve this is to accumulate the signatures of all parties involved. This yields the protocol from Figure 6.

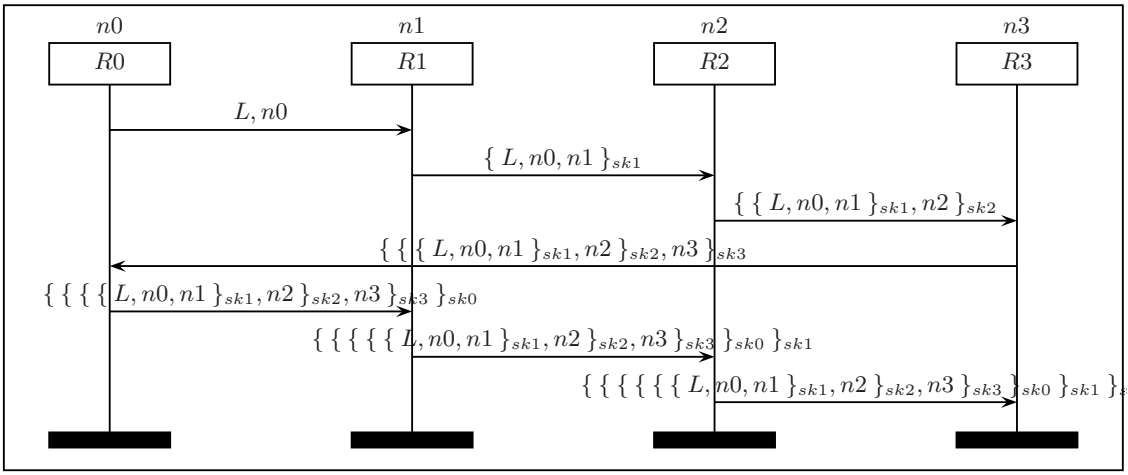

Fig. 6. Four-party NS private-key protocol without delegated authentication, where $L = R0, R1, R2, R3$ (α-nd).

The messages of the generalized protocol are recursively defined as follows.

$$\begin{aligned}
\text{MsgA}_{\alpha\text{-}nd}(0) &= R0, R1, \ldots, R(p-1), n(0) \\
\text{MsgA}_{\alpha\text{-}nd}(i) &= \{\, \text{MsgA}_{\alpha\text{-}nd}(i-1), n(i) \,\}_{sk(Ri)} \quad (\text{for } 0 < i \leq p-1) \\
\text{MsgB}_{\alpha\text{-}nd}(0) &= \{\, \text{MsgA}_{\alpha\text{-}nd}(p-1) \,\}_{sk(R0)} \\
\text{MsgB}_{\alpha\text{-}nd}(i) &= \{\, \text{MsgB}_{\alpha\text{-}nd}(i-1) \,\}_{sk(Ri)} \quad (\text{for } 0 < i \leq p-2)
\end{aligned}$$

Like the previous protocols, this protocol satisfies injective synchronisation. However, in addition it satisfies the fact that attacks on one agent cannot propagate through the protocol. When running a session with one or more compromised parties, we cannot conclude anything about the behaviour of these compromised parties. Because the adversary is able to forge the signatures of the compromised parties they don't even have to satisfy the, rather weak, property of *aliveness*. Therefore, we can only conclude about the status of the honest participants to the session. In particular, we can conclude that all *uncompromised* parties in a session achieve *agreement* over all nonces of these uncompromised parties. For the definition of agreement, we refer to [32].

3.5 Generalized NSL public-key protocol without delegated authentication (β-nd).

As the previous protocol is based on signatures, it does not provide secrecy of the nonces. This can be achieved by merging this protocol with the generalized NSL public-key protocol from Section 3.2. The resulting protocol is illustrated in Figure 7. The messages of this protocol are defined as follows.

$$\begin{aligned}
\text{MsgA}_{\beta\text{-}nd}(i) &= \{\, \text{MsgA}_{\alpha\text{-}nd} \,\}_{pk(next(i))} \\
\text{MsgB}_{\beta\text{-}nd}(i) &= \{\, \text{MsgB}_{\alpha\text{-}nd} \,\}_{pk(next(i))}
\end{aligned}$$

The virtue of this protocol is that it satisfies injective synchronisation and secrecy of the nonces if all participants to a session are uncompromised. In case of compromised participants, it satisfies agreement between uncompromised parties on the nonces of these uncompromised parties. Please notice that we do not have secrecy of the nonces if one or more participants in a session are compromised.

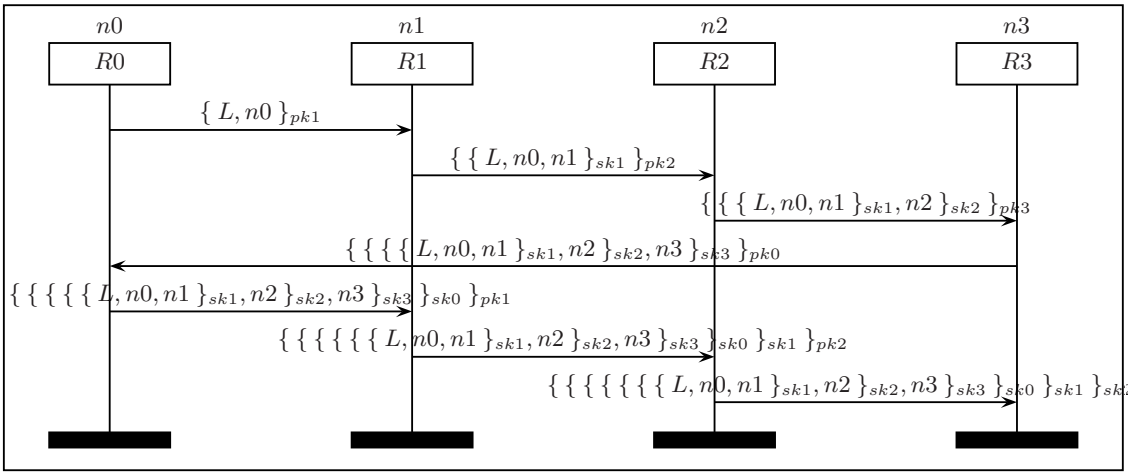

Fig. 7. Four-party NSL public-key protocol without delegated authenication, where $L = R0, R1, R2, R3$ (β-nd).

This property would contradict the requirement that at the end of the protocol all participants know all nonces. It is not possible for an agent to find out whether another agent has been compromised or not, so the intruder will always learn all information made available to the compromised agent.

3.6 Generalized BKE protocol without delegated authentication (β-nd^*).

The generalized BKE protocol without delegated authentication is constructed from the previous protocol by replacing the asymmetric encryption in type B messages by encryption with the recipient's nonce. We will only provide the message definition of this protocol.

$$\text{MsgA}_{\beta\text{-}nd^*}(i) = \{\ \text{MsgA}_{\alpha\text{-}nd}\ \}_{pk(next(i))}$$
$$\text{MsgB}_{\beta\text{-}nd^*}(i) = \{\ \text{MsgB}_{\alpha\text{-}nd}\ \}_{n(i+1)}$$

3.7 Type-flaw attacks.

In assessing the correctness of the six protocols above, we have assumed that type-flaw attacks are not possible, i.e. agents can verify whether an incoming message is correctly typed. There are several reasons for doing this. Without this assumption, there are type-flaw attacks on all members of the defined protocol family provided here. There are not only simple attacks for specific instances of p, but also multi-protocol type-flaw attacks involving instances for several choices of p in one attack, as in [17]. Thus, we find that typing is crucial. Solutions for preventing type-flaw attacks using type information is examined in detail in e.g. [26]. Such type information can be easily added to each message, but a simple labeling will also suffice. If we add a tuple (p, l) before each message inside the encryption, where p is the number of participants for this instance, and l is the label of the message, the protocols become robust against type-flaw attacks and multi-protocol attacks with other instances of itself.

Using an automatic protocol verification tool (Scyther, [16]) we have established that the type-flaw attacks on synchronisation are not due to the specific ordering of the nonces and agent names within the messages. In particular, we examined different options for the message contents (without adding labels): reversing the order of either the agent or the nonce list, interleaving the lists, etc. We established the existence of type-flaw attacks for some choices of p for all variants we constructed.

4 Related Work

In Dolev-Yao style analysis of security protocols, where black-box abstractions of cryptographic operators are considered, protocols usually consist of two or three roles only. There are many recent successful methodologies [23,41] and analysis tools [3,6,15,16] that can be used to analyse protocols in the Dolev-Yao model. All the tools mentioned assume the protocols have a fixed number of participants. Therefore, they cannot be used to analyze multi-party protocols in general, but they can be used to analyze specific instances of such protocols. For example, Proverif [6] has been used to analyze instances of the GDH protocols from [2], and here we have used Scyther [16] to analyze instances of our protocol.

In spite of the success of these methods, few multi-party protocols have been constructed in the Dolev-Yao setting. As a notable exception we would like to mention [12], where the authors construct a challenge-response protocol for any number of parties. However, the protocol described there does not satisfy synchronisation or agreement, which is shown in our technical report [19].

On the other hand, many multi-party protocols have been constructed and analyzed in a cryptographic setting, e.g. [1,2,4,5,7–9,13,27,29,30,34]. These protocols are typically assumed to employ a multicast primitive, and based on this primitive their complexity can be analyzed, as in e.g. [28,33]. Unfortunately the protocols in this category are designed to meet different goals than the protocol presented here, and therefore cannot be used to compare with our approach.

On the borderline between the cryptographic approach and the formal Dolev-Yao approach, the Bull protocol from [11] provides an interesting example. This protocol is flawed, as shown in [38], although a more abstract version was shown to be correct in [36].[1] Unlike the generalized NSL protocol, this protocol is based on agents sharing a symmetric key with a server, and furthermore the server is involved in each session.

Recently, a corpus of multi-party protocols have been established as part of the Coral project [40], aiming to establish a reference set for multi-party protocol analysis.

Regarding proving authentication protocols correct, there have been some recent attempts to simplify such proofs. For example, one successful approach is to use static analysis of the protocol to prove authentication properties, as described in e.g. [10]. However, the notions of authentication used there are weaker than synchronisation or agreement, and the methods used there do not seem suitable for proving synchronisation.

5 Conclusions and Future Work

We proposed a family of security protocols for multi-party authentication and discussed six novel protocols from this family. The first three protocols assume that all agents taking part in the protocol session are uncompromised, which makes it possible to use delegated authentication. In particular, we provided a protocol α which is based on private key encryption (signing), and two protocols β and β^* based on public key encryption, that also provide secrecy of the nonces. Interestingly, the two-party version of β coincides with the Needham-Schroeder-Lowe protocol, and the two party version of β^* coincides with the Bilateral Key Exchange protocol.

The next three protocols α-nd, β-nd, and β-nd^* are strengthened versions of the first three protocols. They use direct authentication instead of delegated authentication, at the cost of a loss in efficiency. Delegated authentication implies that a compromised participant in a protocol session can

[1] In this particular case, the cryptographic implementation differs in a significant way from the abstract version, which allows for new attacks: the exclusive or operator does not satisfy the properties required for black-box encryption, because under certain conditions it is possible to retrieve the key encrypted term.

falsely authenticate other participants of that session. This property has hardly been studied before, since it has no meaning in the common setting of two-party protocols. There it is assumed that although there may be compromised parties, an actual session only has to provide authentication if all (both) partners are uncompromised. A recent study discussing a related issue for secrecy is in [24].

All six protocols satisfy injective synchronisation. Except for the two protocols α and α-nd, that are based solely on private keys, the protocols satisfy secrecy of the nonces as well. The third protocol β^* and its strengthening β-nd^* are optimized with respect to the use of symmetric encryption.

The correctness proofs, although not presented here, are formulated in terms of the operational semantics framework introduced in [18, 20–22] and do not require a fixed number of parties p. This is in line with more recent attempts (e.g. [40]) to develop methodologies for such (parameterised) multi-party protocols, for which these protocols could be used as a case study. Correctness of the protocols is subject to the assumption that the messages include enough information as to allow a receiving agent to check if a message is correctly typed, and correctly split into subterms.

As has been shown by history, constructing correct security protocols is not trivial. Even knowing this, we were surprised to find that all variants of the proposed protocol (irrespective of the ordering of nonces and role names in the messages) suffer from type-flaw attacks. We found this out by using the Scyther tool [16]. In fact, we extensively used this tool to investigate instances of the protocols for a specific number of participants to guide us in our research. A simple (and standard) extension of the messages will make the protocols resilient against such type-flaw attacks.

We showed that the communication structure underlying the protocols can serve as a generic pattern for multi-party challenge-response mechanisms, in which many variants can be captured. These generalized protocols can serve as efficient communication structures underlying multi-party authentication schemes as used in electronic commerce protocols. An interesting topic for future research is to study other members of the protocol family.

References

1. G. Ateniese, M. Steiner, and G. Tsudik. Authenticated group key agreement and friends. In *ACM Conference on Computer and Communications Security*, pages 17–26, 1998.
2. G. Ateniese, M. Steiner, and G. Tsudik. New multiparty authentication services and key agreement protocols. *IEEE Journal on Selected Areas in Communications*, 18(4):628–639, 2000.
3. D. Basin, S. Mödersheim, and L. Viganò. OFMC: A symbolic model checker for security protocols. *International Journal of Information Security*, 4:181–208, 2005.
4. K. Becker and U. Wille. Communication complexity of group key distribution. In *CCS '98: Proceedings of the 5th ACM conference on Computer and communications security*, pages 1–6, New York, NY, USA, 1998. ACM Press.
5. M. Bellare and P. Rogaway. Provably secure session key distribution: the three party case. pages 57–66, 1995.
6. B. Blanchet. An efficient cryptographic protocol verifier based on prolog rules, 2001.
7. C. Boyd and C. Mathuria. *Protocols for Authentication and Key Establishment*. Springer, 2003.
8. C. Boyd and J. Nieto. Round-optimal contributory conference key agreement, 2003.
9. E. Bresson, O. Chevassut, D. Pointcheval, and J.J. Quisquater. Provably authenticated group Diffie-Hellman key exchange. In *CCS '01: Proceedings of the 8th ACM conference on Computer and Communications Security*, pages 255–264, New York, NY, USA, 2001. ACM Press.
10. M. Bugliesi, R. Focardi, and M. Maffei. Authenticity by tagging and typing. In *FMSE '04: Proceedings of the 2004 ACM workshop on Formal methods in security engineering*, pages 1–12, New York, NY, USA, 2004. ACM Press.
11. J. Bull and D. Otway. A nested mutual authentication protocol. *Operating Systems Review*, 33(4):42–47, 1999.
12. L. Buttyán, A. Nagy, and I. Vajda. Efficient multi-party challenge-response protocols for entity authentication. *Periodica Polytechnica*, 45(1):43–64, April 2001.
13. P.C. Cheng and V.D. Gligor. On the formal specification and verification of a multiparty session protocol. In *IEEE Symposium on Security and Privacy*, pages 216–233, 1990.
14. J.A. Clark and J.L. Jacob. A survey of authentication protocol literature. Technical Report 1.0, 1997. `http://citeseer.ist.psu.edu/clark97survey.html`.

15. R. Corin and S. Etalle. An improved constraint-based system for the verification of security protocols. In M. V. Hermenegildo and G. Puebla, editors, *9th Int. Static Analysis Symp. (SAS)*, volume LNCS 2477, pages 326–341, Madrid, Spain, Sep 2002. Springer-Verlag, Berlin.
16. C.J.F. Cremers. The scyther tool: Automatic verification of security protocols. `http://www.win.tue.nl/~ccremers/scyther/`.
17. C.J.F. Cremers. Feasibility of multi-protocol attacks. In *Proc. of The First International Conference on Availability, Reliability and Security (ARES)*, Vienna, Austria, April 2006. IEEE Computer Society Press. To appear.
18. C.J.F. Cremers and S. Mauw. Operational semantics of security protocols. In S. Leue and T. Systä, editors, *Scenarios: Models, Transformations and Tools, International Workshop, Dagstuhl Castle, Germany, September 7-12, 2003, Revised Selected Papers*, volume 3466 of *LNCS*. Springer, 2005.
19. C.J.F. Cremers and S. Mauw. Generalizing Needham-Schroeder-Lowe for multi-party authentication. CS-Report 06/04, Department of Mathematics and Computing Science, Eindhoven University of Technology, 2006.
20. C.J.F. Cremers, S. Mauw, and E.P. de Vink. Defining authentication in a trace model. In Theo Dimitrakos and Fabio Martinelli, editors, *FAST 2003*, pages 131–145, Pisa, September 2003. IITT-CNR technical report.
21. C.J.F. Cremers, S. Mauw, and E.P. de Vink. A syntactic criterion for injectivity of authentication protocols. In P. Degano and L. Vigano, editors, *Arspa 2005*, volume 135(1) of *ENTCS*, pages 23–38, July 2005.
22. C.J.F. Cremers, S. Mauw, and E.P. de Vink. Injective synchronisation: an extension of the authentication hierarchy. *Theoretical Computer Science*, 2006. Special issue on ARSPA'05, (P. Degano and L. Vigano, eds.). To appear.
23. A. Datta, A. Derek, J.C. Mitchell, and D. Pavlovic. Secure protocol composition. In *FMSE '03: Proceedings of the 2003 ACM workshop on Formal methods in security engineering*, pages 11–23, New York, NY, USA, 2003. ACM Press.
24. A. D. Gordon and A. S. A. Jeffrey. Secrecy despite compromise: Types, cryptography, and the pi-calculus. In *Proc. Concur*, Lecture Notes in Computer Science, pages 186–201. Springer-Verlag, 2005.
25. C. He and J.C. Mitchell. Analysis of the 802.11i 4-way handshake. In *WiSe '04: Proceedings of the 2004 ACM workshop on Wireless security*, pages 43–50, New York, NY, USA, 2004. ACM Press.
26. J. Heather, G. Lowe, and S. Schneider. How to prevent type flaw attacks on security protocols. *Journal of Computer Security*, 11(2):217–244, 2003.
27. Just and Vaudenay. Authenticated multi-party key agreement. In *ASIACRYPT: Advances in Cryptology – ASIACRYPT: International Conference on the Theory and Application of Cryptology*. LNCS, Springer, 1996.
28. J. Katz and M. Yung. Scalable protocols for authenticated group key exchange. pages 110–125. LNCS 2729, Springer, 2003.
29. H. Lee, H. Lee, and Y. Lee. Multi-party authenticated key agreement protocols from multilinear forms, 2002.
30. N.Y. Lee and M.F. Lee. Comments on multiparty key exchange scheme. *SIGOPS Oper. Syst. Rev.*, 38(4):70–73, 2004.
31. G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Proceedings of TACAS*, volume 1055, pages 147–166. Springer Verlag, 1996.
32. G. Lowe. A hierarchy of authentication specifications. In *Proc. CSFW '97, Rockport*, pages 31–44. IEEE, 1997.
33. D. Micciancio and S. Panjwani. Optimal communication complexity of generic multicast key distribution. In Jan Camenisch and Christian Cachin, editors, *Advances in cryptology - EUROCRYPT 2004, proceedings of the internarional conference on the theory and application of cryptographic techniques*, volume 3027 of *Lecture Notes in Computer Science*, pages 153–170, Interlaken, Switzerland, May 2004. Springer-Verlag.
34. D. Nalla and K. Reddy. Id-based tripartite authenticated key agreement protocols from pairings, 2003.
35. R. Needham and M. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(2):120–126, February 1978.
36. L.C. Paulson. Mechanized proofs for a recursive authentication protocol. In *10th Computer Security Foundations Workshop*, pages 84–95. IEEE Computer Society Press, 1997.
37. A.W. Roscoe. Intensional Specifications of Security Protocols. In *Proc. CSFW '96*, pages 28–38. IEEE, 1996.
38. P. Y. A. Ryan and S. A. Schneider. An attack on a recursive authentication protocol. a cautionary tale. *Inf. Process. Lett.*, 65(1):7–10, 1998.
39. Security protocols open repository (SPORE). `http://www.lsv.ens-cachan.fr/spore`.
40. G. Steel. Coral project: Group protocol corpus, 2004. `http://homepages.inf.ed.ac.uk/gsteel/group-protocol-corpus`.
41. F.J. Thayer Fábrega, J.C. Herzog, and J.D. Guttman. Strand spaces: Why is a security protocol correct? In *Proc. 1998 IEEE Symposium on Security and Privacy*, pages 66–77, Oakland, California, 1998.