Names in Cryptographic Protocols*

Simone Lupetti, Feike W. Dillema and Tage Stabell-Kulø

Department of Computer Science, University of Tromsø, Norway

Abstract. This paper discusses the security properties of names. To fulfill their role in cryptographic protocols, names must be unique across correlated sessions i.e. where the massages of one session can be reused in another without detection and that uniqueness must be guaranteed to hold for each participant of these runs. To provide and verify uniqueness, two different mechanisms are shown possible, namely local and global verification. In both cases we discuss the implications of uniqueness on the execution environment of a cryptographic protocol, pointing out the inescapable issues related to each of the two mechanisms. We argue that such implications should be given careful consideration as they represent important elements in the evaluation of a cryptographic protocol itself.

1 Introduction

The number of messages and message elements that are relevant for the security of a protocol is normally rather small. Message elements encountered are keys, nonces, timestamps, identifiers and, occasionally, a few others.

Assumptions must be made regarding the properties of these elements, the participants of the protocol and their trustworthiness, their abilities, the properties of the communication infrastructure, and the environment in which the protocol is deployed [1]. While assumptions directly related to the functioning of the protocol are almost always made explicit and discussed at large, assumptions on the running environment are, more often than not, left unspecified [2]. One consequence of this is that the cost factors considered when evaluating a cryptographic protocol are usually limited to conceptual simplicity, encryption, length, and number of messages [3, p. 34]. External factors such as requiring secure synchronized clocks, synchronous communication or broadcast communication channels are less often considered as integral part of the protocol security [4].

We focus here on the single issue of *names* by analyzing their security requirements. Abadi et al. raised the issue of names in cryptographic protocols by formulating a specific prudent engineering principle:

> *"If the identity of a principal is essential to the meaning of a message, it is prudent to mention the principal's name explicitly in the message."*[5]

* This work has been generously supported by the Norwegian Research Council through the projects Arctic Bean (project number 146986/431) and PENNE (project number 158596/431).

However, like other message elements, names have properties that must be upheld to justify their use. Using some well-known cryptographic protocols as a starting point, we state that to fulfill their role in cryptographic protocols, names must be unique across sessions of the same protocol that are *correlated* to each other, meaning that they are executed in a time window where replay attacks are possible. We present two distinct methods to fulfill such a requirement and discuss their peculiarities. In both cases we show how uniqueness of names is a source of external costs and we point out the practical implications of this for the design and evaluation of cryptographic protocols.

In the next section, we recall examples of protocol attacks that can be fixed by the proper inclusion of names. These examples are used in Sec. 3 to present two possible ways to provide verifiable unique names. In Sec. 4 we discuss the property of uniqueness and its implications on the name system. We conclude in Sec. 5.

2 Examples

We use the same set of examples as [5, Sec. 4] due to their variety. We give only a minimal description of these here and refer the reader to the original article for more details about the attacks, the fixes and the used notation.

Example 1 The first example is a key exchange protocol using asymmetric encryption by Denning and Sacco [6, p. 535]:

$$\begin{array}{l} \textit{Msg } 1\ A \rightarrow B : A, B \\ \textit{Msg } 2\ B \rightarrow A : C_A, C_B \\ \textit{Msg } 3\ A \rightarrow B : C_A, C_B, \{\{K_{ab}, T_a\}_{K_a^{-1}}\}_{K_b} \end{array}$$

A and B represent the names of the two principals participating in the protocol: Alice and Bob. The certificates C_A and C_A bind these two principals to their respective public keys K_a and K_b. T_a is a timestamp and K_{ab} is the session key being exchanged. The problem is that once Bob receives the third message, he can remove the outer encryption layer and re-encrypt the result for an outsider Charlie. In particular, Bob could send:

$$\textit{Msg } 3'\ B \rightarrow C : C_A, C_C, \{\{K_{ab}, T_a\}_{K_a^{-1}}\}_{K_c}$$

At this point, and for the duration of the validity of the time stamp T_a, Bob can impersonate Alice to Charlie. The proposed fix is that the third message should contain at least the name of Bob in the inner encrypted section[1]:

$$\textit{Msg } 3''\ B \rightarrow C : C_A, C_B, \{\{B, K_{ab}, T_a\}_{K_a^{-1}}\}_{K_b}$$

It is possible now for Charlie to understand that this message is not part of his run of the protocol but was destined for B instead.

[1] In the original article both Alice's and Bob's names appear in the fix but it is also pointed out that Alice's name can be deducted from K_a^{-1}. We then omit it here.

Example 2 A similar flaw is present in the following authentication protocol using symmetric key encryption by Woo and Lam [7, pp. 42-43]:

$$\begin{array}{l} \textit{Msg } 1\ A \rightarrow B : A \\ \textit{Msg } 2\ B \rightarrow A : N_b \\ \textit{Msg } 3\ A \rightarrow B : \{N_b\}_{K_{as}} \\ \textit{Msg } 4\ B \rightarrow S : \{A, \{N_b\}_{K_{as}}\}_{K_{bs}} \\ \textit{Msg } 5\ S \rightarrow B : \{N_b\}_{K_{bs}} \end{array}$$

Where K_{as} and K_{bs} are two keys that the server S shares with Alice and Bob, respectively and N_b is a nonce generated by Bob. The purpose of this protocol is to allow Bob to check whether Alice is on-line. This protocol is subject to an attack where a third party Charlie impersonates Alice to Bob. Briefly, if Charlie presents itself to Bob both as Charlie and Alice at the same time, Bob is tricked to believe that Alice is present while she is not. The proposed fix again involves the use of names. In this case it requires the last message to include Alice's name:

$$\textit{Msg } 5'\ S \rightarrow B : \{A, N_b\}_{K_{bs}}$$

Example 3 The last example protocol is an early version of SSL [8], in which Bob can impersonate Alice to a third party Charlie:

$$\begin{array}{l} \textit{Msg } 1\ A \rightarrow B : \{K_{ab}\}_{K_b} \\ \textit{Msg } 2\ B \rightarrow A : \{N_b\}_{K_{ab}} \\ \textit{Msg } 3\ A \rightarrow B : \{C_A, \{N_b\}_{K_a^{-1}}\}_{K_{ab}} \end{array}$$

K_{ab} is the session key to be used between Alice and Bob, K_b is Bob's public key and N_b is a nonce. The proposed fix is to insert Bob's and Alice's names in the final message[2]:

$$\textit{Msg } 3'\ A \rightarrow B : \{C_A, \{B, N_b\}_{K_a^{-1}}\}_{K_{ab}}$$

□

While all these fixes seem straightforward, some questions may arise regarding their implications: What properties, if any, must be provided in order for a name to be used in the security protocol? How can these properties be upheld? Does this impose requirements on the environment in which the protocol is deployed and on the naming system in use? We investigate these questions in the next section with the help of the previous examples.

3 Names

The primary role of names is that they let one reference a resource, allowing sharing. In the presented examples, however, names are used as a mean to identify a specific protocol run by identifying the protocol participants.

[2] As for Example 1 earlier, we omit Alice's name because this can be deduced form her signature.

Scalable naming systems designed with sharing in mind usually allow for short-term inconsistencies to achieve availability and performance. The Domain Name System (DNS) [9] is an example of such a naming system; domain name entries are replicated and can be cached freely on any server and client for performance. This makes the naming system scalable and responsive in the light of unreliable message delivery, failures, and network partitions but it leaves to the users to check whether a referred resource is the correct one. This means that the DNS is subject to spoofing attacks *by design*. In other words, naming systems designed for resource sharing like the DNS are unsuitable for use in security protocols not being designed for identification purposes.

The question is then what properties names must have to be usable in security protocols. Do we have to remove all potential for naming inconsistencies and sacrifice availability, performance (and thus scalability) for security? The following principle states what must be achieved:

> *When the security of a protocol relies on the identity of a principal, the uniqueness of principals' names across all correlated sessions of the same protocol is required.*

Two or more protocol runs are correlated when executed within a time window where messages of one can be reused in another without detection. This window is determinate by the semantic of the protocol. For the first protocol for example, this window is bounded by (the semantics of) the timestamp T_a in the third message. In Example 2 and 3 instead, since the presented attacks require that the attacker interleaves the messages of its session with the ones of the session he intends to attack, this window is determined by the temporal length of the attacked session.

This principle has been distilled from the previous examples. From them it is straightforward to extrapolate the uniqueness of participant names as the necessary condition for the proposed fixes to hold. For example, with reference to the first protocol, if Bob's and Charlie's names B and C are identical or can be confused with each other (maybe just at bit-string level) the proposed fix does not help anymore to fend off the attack. The same is true also in both the second and the third example. Uniqueness, on the other hand, is not needed outside the scope of the correlated instances of a specific protocol run since (assuming that messages belonging to different protocols are distinguishable as such) illegitimate messages belonging to another sessions of the same protocol are here detected by other means.

3.1 Guaranteeing Uniqueness

In addition to the ability to generate unique names, it must be safe for the principals of a security protocol to assume that this uniqueness is maintained even in light of malicious partecipants. In other words, it must be infeasible for a malicious principal to violate the uniqueness assumption as the basis for an attack. Since uniqueness of names is, in general, required only across different sessions of the same protocol, a principal has two possible ways for verifying this:

Local Verification. One or more of the participants in a protocol run verify locally the uniqueness of the names used in that run. Local verification means here that the uniqueness check only involves local state and messages belonging to the protocol run itself, not entailing additional communication with third parties. Correlated sessions using the same names are prohibited and serialized as a consequence. In this solution the integrity of the namespace does not need to be protected since a name is *verified* to be unique every time it is used.

Global Verification. A trusted naming system can be relied upon to be resilient to tampering. Such an infrastructure is global and guarantees the integrity and the consistency of the namespace providing unique names. In this case, an authority hands out unforgeable and tamper-proof names for all the principals in the system.

To sum up, the consistency of the namespace must be either enforced globally or verified locally. Local and global verification each have, as we will show, their own applicability range that is delimited both by the protocol in question and by environmental constraints.

3.2 Local Verification and Verifiers

Not all principals participating in a protocol run are interested in (verifying) the uniqueness of names. In the previous examples (and arguably in the general case), the principal that is concerned about the uniqueness of names is the potential victim of the attack that is countered by the use of names. Obviously, this role is protocol-dependent. Therefore a variety of scenarios is possible when local verification is used.We discuss some of them using the example protocols we presented earlier.

Example 1 To avoid the reply attack presented in Example 1, Bob's name is added to the message being replayed in the attack. This name must be verified to be unique in order to thwart the attack. Charlie has no way, from the messages he receives from Alice, to check whether his and Bob's names are identical and in use in correlated protocol runs by Alice (hence creating the precondition for a replay attack). Direct verification of the names by Charlie is therefore not possible. Nonetheless he could trust Alice to perform the name check on his behalf. Alice, when opening two correlated executions of the protocol with Bob and Charlie, is able to check whether the names B and C (to be) sent in the messages of the two protocol runs are the same or not:

$$\begin{array}{ccc} \textit{Msg}\ 1\ A \rightarrow B : A, B & \qquad & \textit{Msg}\ 1'\ A \rightarrow C : A, C \\ \vdots & & \vdots \end{array}$$

If Alice detects such duplicate names used in correlated protocol runs, she can protect Charlie by aborting one or both of the protocol runs. Notice that Alice is already part of Charlie's Trusted Computing Base (TCB) since Charlie already trusts her for other security-critical tasks like picking a good K_{ab} and for not disclosing it to third parties, for example. It is then reasonable in this case for him to also rely on Alice for protection against this replay attack.

To sum up, local verification is possible in this protocol under the condition that the potential victim of the attack, Charlie, trusts the initiator of the protocol, Alice, to verify the uniqueness of the names of his counterparts.

Example 2 Local verification is possible in this protocol also: For the attack to be mounted Charlie initiates two concurrent sessions of the protocol with the victim Bob. In these sessions, Charlie calls himself by two different names; his own and the name of the principal that he is impersonating:

$$\begin{array}{lcl} \textit{Msg}\ 1\ C \rightarrow B : A & \qquad & \textit{Msg}\ 1'\ C \rightarrow B : C \\ \quad\vdots & & \quad\vdots \end{array}$$

Bob registers the two names, A and C, and uses them ad verbatim in the messages of the protocol he sends later. This allows him to verify whether the names used in the different sessions are the same or not. In other words, the designated victim of the replay attack is able here to verify itself whether the names used in correlated protocol sessions are unique.

Example 3 Our third and final example gives us the opportunity to discuss a scenario where local verification by one or more protocol participants is not possible. We report the replay attack on the protocol at full because it was left as an exercise to the reader in the original paper:

$$\begin{array}{ll} \textit{Msg}\ 1 & A \rightarrow B : \{K_{ab}\}_{K_b} \\ \textit{Msg}\ 1' & B \rightarrow C : \{K_{ab}\}_{K_c} \\ \textit{Msg}\ 2' & C \rightarrow B : \{N_b\}_{K_{ab}} \\ \textit{Msg}\ 2 & B \rightarrow A : \{N_b\}_{K_{ab}} \\ \textit{Msg}\ 3 & A \rightarrow B : \{C_A, \{N_b\}_{K_a^{-1}}\}_{K_{ab}} \\ \textit{Msg}\ 3' & B \rightarrow C : \{C_A, \{N_b\}_{K_a^{-1}}\}_{K_{ab}} \end{array}$$

In this protocol Charlie, the victim of the attack, has no way to verify the uniqueness of names from the messages he receives. Moreover, he cannot rely on his counterpart to perform such a check as was possible in Example 1. After all, that counterpart is the principal mounting the attack. Hence, local verification is not possible for this type of protocol and global verification has to be used to guarantee uniqueness.

3.3 Global Verification

The most reliable way to enforce the integrity and consistency of a namespace is to use digital identity certificates. These are issued and digitally signed by a certification authority (CA), on behalf of all the principals in the system binding each of them to its public key. All of this, along with mechanisms for verifying the validity and revoking certificates constitutes a Public Key Infrastructure (PKI). This can provide for a reliable and trusted naming system. When the integrity and the consistency of the namespace is provided using a PKI, one could then replace the names used in the protocol with identity certificates for the named principals. This means:

$$\textit{Msg } 3'' \ B \to C : C_A, C_B, \{\{C_B, K_{ab}, T_a\}_{K_a^{-1}}\}_{K_b}$$

for the first protocol, and:

$$\textit{Msg } 3'' \ A \to B : \{C_A, \{C_B, N_b\}_{K_a^{-1}}\}_{K_{ab}}$$

for the third one. Alternatively, one could use Alice's and Bob's public keys K_a and K_b (or their hashes) in place of their names. This is a common practice in the implementation of cryptographic protocols, but we recommend also to replace references (in subscripts) to the name B with references to the public key K_b already in the protocol description. This clarifies the properties we require of the names. To this end, however, rewriting the protocol to use certificates for principals' names is even better than when simply public keys are used in their place. Using certificates makes explicit in the protocol description the necessity of an authority taking responsibility for the integrity of the namespace.

4 Discussion

As opposite as for naming systems that are specifically designed for sharing of resources (like the DNS), a namespace whose main requirements are to be consistent and non tamperable must sacrifice performance, scalability and availability because of its dependencies to a PKI [10]. Moreover, by relying on a PKI, the system inherits all its open problems like privacy concerns (in setting where they are relevant) and revocation-related issues [11, 12].

However, in cases where public key cryptography is already required (Examples 1 and 3), such a solution may be preferable since it comes at zero additional cost in that the costs for deploying the PKI have been already sustained. Also, as already shown, local verification is not always possible, forcing in these cases to resort on a global and consistent namespace.

On the other hand, local verification does not rely on distributed state and does not add dependencies between distributed parties. However, even when local verification is possible it has its own share of limitations. Local verification requires principals to keep track of all names used in their correlated protocol sessions. All concurrent executions of the same protocol are also correlated if session identifiers are not used. Threading is a popular way to implement concurrently running instances of a protocol, and the list of names in use would be a resource shared between all threads requiring synchronized access. This list does not only grow linearly with the number of concurrent protocol sessions, but it may create a bottleneck because it is shared between all threads. Notice also, that name caching can not be used as would be possible when global verification is used.

Last but certainly not least, local verification is dependent on what principal in the protocol represents the attacker and what principal represents the verifier. This makes local verification dependent on the protocol itself. An *a priori* analysis of the protocol must be performed then to assure the feasibility and the correctness of the verification process. Such an analysis can only be performed for known replay attacks. Unlike a global verification, local verification can not therefore be used as as a *prevention* mechanism but only as a remedy for known exploits. In contrast, global verification provides global integrity for names, such that these can be employed also as a precautionary measure for all protocols countering attacks that are both known and not known to the implementor of the protocol.

4.1 Local vs. Global Uniqueness

We have argued that the peculiar, but not unusual, use of names in cryptographic protocols makes system-wide uniqueness not necessary in all cases. Global uniqueness in fact, while not strictly required by the cryptographic protocol, may be convenient because the same global naming system is already in use to identify all principals in the system. This is the argument that leads, for example, to the adoption of public keys (and as consequence of a PKI) in places of names.

Uniqueness of names becomes an issue also when the name system implements revocation and re-allocation of names. For example, in IPSEC, IP addresses are used to name communication endpoints [13]. When these are dynamically assigned (using the DHCP protocol, for example), protocol affiliation of messages should be extended also to subsequent sessions. Otherwise, a message intended for Bob, hence containing his name B, could be used in a replay attack against Charlie once Bob's name (IP address) has been reassigned to Charlie. Uniqueness of names may have to be maintained therefore also over time, for sessions that are not correlated. The use of session identifiers removes the reliance on (the uniqueness of) the names of protocol participants but, while naming sessions explicitly is effective, the integrity of session identifiers must be provided as well, possibly implying a substantial modification of the original protocol.

The alternative to name recycling is to use lingering names across subsequent sessions. Such names are usually referred to as *persistent* or *inescapable* [14]. To implement such names, however, may be a non-trivial exercise for a systems designer. For example, the namespace used must be "big enough" to never generate colliding names for different principals. A design flaw or implementation bug allowing a wrap-around of the name space may have serious and direct security implications. Determining how big is "big enough" is complicated by the fact that this is not merely determined by the worst-case usage rate of names in the system, but also by the worst-case abuse rate. In other words, unless the rate of name consumption is bounded somehow, a determined attacker is able to exhaust a name space independently of how big it is [15].

4.2 Design Alternatives

All of the presented attacks require a principal to run two or more correlated executions of the same protocol with different counterparts. A message belonging to one session can be maliciously used this way in another (and *vice-versa*). In these cases, the attack can be thwarted by serializing all executions of the protocol at the victim. Note that, according our definition of correlation, to just execute a protocol after another may not be enough if the windows of vulnerability extends also after a run is finished (as in Example 1). In the general case, to serialize therefore means to execute only one protocol during each of this periods.

A possible solution to fend off replay attacks is then to restrict the execution environment of the protocol and so forbid multiple correlated executions of the protocol by the same principals. Of course, this kind of restriction on the execution environment may negatively affect the system's performance, e.g. its responsiveness and maximum throughput.

Some would argue that relying on the execution environment of the protocol for protection to replay attacks is inferior from adding protection to the cryptographic protocol itself by adding names. However, names to be used in a cryptographic protocol must be secured either by the local implementation or by an external infrastructure. This places both solutions on equal footing, or some might even argue that a solution that does not require a change in the protocol is even preferable. Solutions against replay attacks do not merely rely on the design of the cryptographic protocol, but always rely on external support. The key point is then what are the different externalities and the economic and technical cost associated to each solution.

5 Conclusions

Names used in authentication protocols must enjoy uniqueness across correlated sessions of the same protocol to be effective. This property is assumed to be provided in the execution environment of the protocol and must be verifiable by its participants. For this, a system can either use a global naming system that provides unforgeable names to every principal in the system or resort to local verification by the concerned parties only.

In the case of a trusted naming system, this infrastructure becomes part of the trusted computing base of each principal of the protocol. The most common solution for such a service is to rely on a PKI and identity certificates. This kind of infrastructures are complex, costly to deploy and manage, and not risk-free [16].

On the other hand, local verification of uniqueness of principal's names is not always possible. Even when it is, local verification is a design choice that depends on the protocol in consideration and does not allow a general "one-for-all" implementation. Reimplementing the solution every time increases the probability of design errors and implementation bugs. In addition, as opposed to the PKI solution, it is not possible to use local verification as a preventive measure but only as a remedy to known attacks.

Each solution has different implications, both technical and economic, associated with it. In either case, these costs may overwhelm the ones traditionally associated with cryptographic protocols such as the costs of communication and encryption. An *a priori* estimation of these costs should be made at design time because these may dominate total costs in the end and be a crucial factor in the (technical and economic) success of the system. Finally, we argued that a system designer should not disregard alternative solutions for protecting from replay attacks.

References

1. Anderson, R., Needham, R.: Robustness principles for public key protocols. In: CRYPTO '95: Proceedings of the 15th Annual International Cryptology Conference on Advances in Cryptology. (1995) 236–247
2. Canetti, R., Meadows, C., Syverson, P.: Environmental requirements for authentication. In: Proceedings of the International Symposium on Software Security. (2002) 339–355
3. Woo, T.Y.C., Lam, S.S.: A lesson on authentication protocol design. Operating Systems Review **28** (1994) 24–37
4. Gong, L.: A security risk of depending on synchronized clocks. Operating Systems Review **26** (1992) 49–53
5. Abadi, M., Needham, R.: Prudent engineering practice for cryptographic protocols. IEEE Transactions on Software Engineering **22** (1996) 6–15
6. Denning, D.E., Sacco, G.M.: Timestamps in key distribution protocols. Commun. ACM **24** (1981) 533–536
7. Woo, T.Y.C., Lam, S.S.: Authentication for distributed systems. In Stallings, W., ed.: Practical Cryptography for Data Internetworks. Volume 25. IEEE Computer Society (1996)
8. Hickman, K.E.B.: The SSL Protocol. RFC, Netscape Communications Corp. (1994)
9. Mockapetris, P.: Domain names - concept and facilities. RFC 1034, The Internet Society (1987)
10. Stabell-Kulø, T., Lupetti, S.: Public-key cryptography and availability. In: Proceedings of SAFECOMP 2005. (2005) 222–232
11. Brand, S.: Rethinking Public Key Infrastructures and Digital Certificates - Building in Privacy. The MIT Press (2000)
12. Millen, J., Wrigh, R.N.: Certificate revocation the responsible way. In: Proceedings of Computer Security, Dependability, and Assurance: From Needs to Solutions (CSDA'98), IEEE Computer Society (1999) 196–203
13. Kent, S., Atkinson, R.: Security architecture for the internet protocol. RFC 2401, The Internet Society (1998)
14. Ellison, C.M., Frantz, B., Lampson, B., Rivest, R., Thomas, B., Ylonen, T.: SPKI certificate theory. RFC 2693, The Internet Society (1999)
15. Douceur, J.R.: The sybil attack. In: Proceedings of the IPTPS02 Workshop. (2002)
16. Ellison, C., Schneier, B.: Ten risks of PKI: What you're not being told about public key infrastructure. Computer Security Journal **16** (2000) 1–7