Soundness Conditions for Cryptographic Algorithms and Parameters Abstractions in Formal Security Protocol Models

Alfredo Pironti, Riccardo Sisto
Politecnico di Torino
Dip. di Automatica e Informatica
c.so Duca degli Abruzzi 24, I-10129 Torino (Italy)
e-mail:{alfredo.pironti, riccardo.sisto}@polito.it
phone: +390115647073 fax: +390115647099

Abstract

When using formal methods, security protocols are usually modeled with a high level of abstraction. In particular, encryption is assumed to be perfect and cryptographic algorithms and their parameters are often abstracted away.

This paper states a set of constraints under which, if an abstract protocol model is secure, then a refined model, which takes into account cryptographic algorithms and parameters, is implied to be secure too. The paper also indicates possible exploitations of this result.

1: Introduction

Several techniques are now available to formally analyze security protocols. They are mostly based on the work by Dolev and Yao [2], where protocol messages are represented as instances of high level abstract data types. Recently, some researchers have started addressing the question of how to get assurance about the fact that the logical correctness of an abstract protocol is indeed preserved when concrete versions of the protocol are defined and when their implementations are developed using programming languages. In general, security faults not present in an abstract protocol specification might be introduced when adding implementation details such as real cryptography and protocol role implementation.

Specifically, some researchers have been working towards ensuring that the formal model used for the analysis of a protocol is a safe abstraction of the protocol implementation, under the assumption that the cryptographic and communication libraries used by the implementation behave as specified by their ideal abstract Dolev-Yao models [2]. In particular, two different strategies have been explored, namely automatic code generation from abstract models ([6, 8]) and automatic model extraction from implementation code ([1, 3]).

This paper presents some results that can be exploited in both approaches, in order to simplify formal analysis: it states sufficient conditions under which the specification of cryptographic algorithms and their parameters can be soundly abstracted away in Dolev-Yao models. First, formal models including the specification of cryptographic algorithms and parameters are defined. Then, these models are simplified, using fault-preserving transformations like the ones introduced in [4]. Under some simple assumptions, it can be shown

that the protocol models including these details can be transformed back into the original abstract protocol models without them, and the classical security faults (secrecy and authentication) are preserved in this transformation.

The remainder of the paper is organized as follows. Section 2 introduces notation and the modelling approach that is used throughout the paper. Section 3 shows how cryptographic algorithms and parameters can be added to Dolev-Yao security protocol models, and specifies the conditions under which they can be abstracted away. Section 4 discusses about the application of the obtained results through an example. Finally, section 5 concludes with an overview of the achievements that have been reached and their practical impact.

2: Abstract Protocol Models and Notation

The formalism used in this paper is based on CSP [7], and the datatype definitions and protocol models are taken from [4, 5]. Essentially, they follow the Dolev-Yao approach.

The datatype used in this work has been developed so that no new types are added in order to represent cryptographic algorithms and parameters, because the idea is to have a single datatype that can be used to model a protocol at different detail levels. $Message$ is the set of all messages of the datatype. For any messages $M, M', K \in Message$, (M, M') denotes a pair, that is a compound message composed of M and M'; $K^{\sim}, K^{+}, K^{-}$ denote respectively a symmetric shared key, an asymmetric public key, and an asymmetric private key, constructed from key material K; $H(M)$ denotes the result of hashing M; $\{M\}_K, \{[M]\}_K, [\{M\}]_K$ denote shared, public, private key encryption of M respectively.

A honest agent can take part in a protocol by using the events:

$send.A.B.M$ if agent A sends message M, with intended recipient B;

$receive.A.B.M$ if agent B receives message M, apparently from agent A;

$claimSecret.A.B.M$ if A thinks that M is a secret shared only with B; if B is not the intruder, then the intruder should not learn M;

$running.A.B.M_s$ if A thinks it is running the protocol with B; M_s is a message sequence, recording some details about the run in question, on which agreement is required.

$finished.A.B.M_s$ if A thinks it has finished a run of the protocol with B; M_s is a message sequence, recording some details about the run in question.

The *send* and *receive* events can also be treated as channels, used by agents to exchange data; the remaining events are used to formally define the desired security properties of the protocol (secrecy and authentication). $Honest$ is the set of all honest agents.

The intruder acts as the medium, thus being allowed to see, modify, forge or drop any message. Its ability to forge new messages from the previously learnt messages is described by a knowledge derivation relation $\vdash$. For instance, if the intruder knows an encryption $\{M\}_{K^{\sim}}$ and the associated key $K^{\sim}$, then by $\vdash$ it can get the plaintext M. The formal definition of the intruder is a process $INTRUDER(S)$ that can interact with any actor on the *send* and *receive* communication channels, and can execute the $leak.M$ event when the intruder can derive M from its current knowledge. The set of all agents is defined as $Agent = Honest \cup \{INTRUDER\}$.

For actors A and B, the abstract formal model of a protocol can be represented as in figure 1. The model representing all the honest agents and the intruder is called $SYSTEM$, and is formally defined as

$$SYSTEM \triangleq (|||_{A \in Honest} P_A) \parallel INTRUDER(IK_0)$$

Figure 1. Actors A and B with $INTRUDER$ in $SYSTEM$.

where, for each $A \in Honest$, P_A is the CSP process that describes A's behavior, IK_0 is the initial intruder knowledge, and $|||$ and $\|$ are the parallel compositions without and with synchronization respectively.

Like in [4], security properties are specified as trace properties. A trace specification $SPEC(tr)$ is a predicate whose free variable tr represents a trace. A process satisfies a specification if the $SPEC(tr)$ predicate is true for all the traces of the process. Two predicates, namely secrecy and injective authentication (or simply authentication), are defined.

Secrecy states that if agent A believes that message M is shared only with honest agent B, then the intruder must not be able to derive M from its knowledge.

$$Secrecy(tr) \triangleq \forall\ A \in Agent; B \in Honest \cdot\ claimSecret.A.B.M \text{ in } tr \Rightarrow \neg leak.M \text{ in } tr$$

Authentication states that, for each protocol run that A thinks it has finished with B, B must have started a protocol run with A, and both A and B must agree on some set M_s of data, belonging to an agreement set $AgSet$.

$$Agreement_{AgSet}(tr) \triangleq \forall\ A \in Agent; B \in Honest; M_s \in AgSet \cdot \\ tr \downarrow finished.A.B.M_s \leq tr \downarrow running.B.A.M_s$$

where $tr \downarrow ev$ is the number of events ev appearing in the trace tr.

3: Modelling and Simplifying Cryptographic Algorithms and Parameters

The Dolev-Yao model used in this work assumes perfect encryption, i.e. cryprographic primitives are represented by means of their ideal properties. However, different real cryptographic algorithms obtain encryption properties in different, incompatible ways. In order to describe this issue, cryptographic algorithms and their parameters are sometimes introduced in abstract descriptions. For example, in [4, 1], encryption functions are distinguished according to the algorithm (e.g. AES or 3DES) and parameters (e.g. CBC or ECB mode) they use.

In order to similarly represent cryptographic algorithms and parameters in our modelling framework, let $Parameter \subseteq Message$ be the set of messages that represent different choices of cryptographic algorithms and parameters. Each protocol logic P_A is then transformed into a refined protocol logic P'_A that takes cryptographic algorithms and parameters into account, leading to a refined $SYSTEM'$, where the intruder is left untouched but its initial knowledge is now called IK'_0, and every abstract protocol logic P_A is replaced by its refined one P'_A. In particular, if $a, b \in Parameter$, in order to get each P'_A from its corresponding P_A, any key K^*, where the symbol * ranges over $\{^\sim, ^+, ^-\}$, that occurs in any event in P_A, is refined as

$$(a, K)^* \tag{1}$$

where a represents key construction parameters (e.g. key type and length). Similarly any encryption $\{M\}_K$ that occurs in any event in P_A, is refined as

$$\{(a, M)\}_K, \tag{2}$$

where a represents the parameters of encryption (e.g. algorithm and related parameters). Same reasoning applies to asymmetric encryptions. In the same way, any hash $H(M)$ that occurs in any event in P_A is refined as

$$H(a, M) \tag{3}$$

It is worth noting that an accurate model must set, for each message M that is refined, its correct cryptographic parameters a according to the protocol specification documents. It can also be noted that the cryptographic parameters may or may not be already present in P_A. For example, if the cryptographic parameters a are being negotiated within the protocol logic, then a will be already present in P_A.

The (1), (2), (3) models correspond to the ones used in [4, 1] where different encryption, decryption and hashing functions are used for each different choice of algorithms and parameters. It is worth noting that this model can fully handle the case where both encryption and key construction have their own, possibly different parameters.

3.1: Simplifying the Cryptographic Algorithms and Parameters Model

Since each P'_A is being built from P_A (plus other information), it is possible to find a simplifying transformation that can take from P'_A back to P_A. Simplifying transformations have been introduced in [4] and extended in [5]. They are briefly recalled here.

A fault-preserving simplifying transformation in its simplest form is a function $f : Message \rightarrow Message$ that defines how messages in the original protocol are replaced by messages in the simplified protocol. The function f is then overloaded to take events, traces and processes, such that all messages in the events, traces or processes are replaced.

Since the refinement procedure proposed here adds parameters $a \in Parameter$ in well known places of particular messages, in order to transform P'_A back into P_A, and thus $SYSTEM'$ into $SYSTEM$, it is possible to define a fault preserving transformation $f(\cdot)$, that removes parameters in well known places. The $f(\cdot)$ function is defined as the identity function, except for the following cases:

$$\begin{aligned}
f(M, M') &= (f(M), f(M')) \\
f(\{(a, M)\}_K) &= \{f(M)\}_{f(K)} \\
f(\{[(a, M)]\}_K) &= \{[f(M)]\}_{f(K)} \\
f([\{(a, M)\}]_K) &= [\{f(M)\}]_{f(K)} \\
f(H((a, M))) &= H(f(M)) \\
f((a, K)^*) &= (f(K))^*
\end{aligned}$$

By the definition of $f(\cdot)$, it follows that $P_A = f(P'_A)$, so a relation between $SYSTEM$ and $SYSTEM'$ has been formally defined. Preservation of secrecy and authentication when $SYSTEM$ is refined into $SYSTEM'$ is formally stated by the following theorems.

Theorem 1

$$IK_0 \supseteq f(IK_0') \cup f(Parameter) \tag{4}$$
$$\Longrightarrow$$
$$SYSTEM \ \textbf{\textit{sat}} \ Secrecy \Rightarrow SYSTEM' \ \textbf{\textit{sat}} \ Secrecy$$

Theorem 2 *If* (4) *and*

$$\forall \ M, M' \in AgSet \ \cdot \ M \sim M' \Rightarrow M = M' \tag{5}$$

hold, then

$$SYSTEM \ \textbf{\textit{sat}} \ Agreement_{f(AgSet)} \Rightarrow SYSTEM' \ \textbf{\textit{sat}} \ Agreement_{AgSet}$$

Their proofs can be carried out like it is done in [5] for similar theorems. Here, the notation $M \sim M'$ means that M and M' can be obtained by applying different cryptographic algorithms and parameters to the same data. So, condition (5) states that $AgSet$ cannot contain two elements that only differ for the cryptographic algorithms or parameters that have been used in building them. In practice, this condition can be enforced by explicitly including, within each message upon which agreement is required, the cryptographic algorithms and parameters that must be used in each part of the message itself. Agreement on the cryptographic algorithms and parameters used for data on which agreement is required is an authentication property that holds if the negotiation algorithm used in the protocol to establish such parameters is logically correct in an unsafe environment. This can be verified as part of the formal protocol verification task on the abstract protocol, as shown in section 4.

Theorem 1 states that cryptographic algorithms and related parameters can be completely abstracted away in secrecy verifications, provided that condition (4) holds, i.e. provided that the intruder is assumed to know cryptographic algorithms and parameters, which is a reasonable assumption.

Then, theorem 2 states that, when verifying authentication, the model of cryptographic algorithms and related parameters can be completely abstracted away if condition (5) holds too, which, as stated above, can be enforced by specifying (and verifying) explicit agreement on all the cryptographic algorithms and parameters used to build data on which agreement is required.

4: Applying the Results to an SSH Transport Layer Protocol Client

In this example, another syntactic sugar is added: lists of n messages are reduced to nested pairs. So, for example, (M, M', M'') is equal to $(M, (M', M''))$.

The SSH Transport Layer Protocol [10] (SSH-TLP) is part of the SSH three protocols suite [9]; in particular it is the first protocol that is used in order to establish an SSH connection between client and server. SSH-TLP gives server authentication to the client, and establishes a set of session shared secrets.

A possible abstract model of an SSH-TLP client is reported in figure 2. The $SSHClient$ process begins a protocol session with the server by sending it the client identification string denoted IDC. The server responds with IDS, the server identification string. Then the

$$
\begin{aligned}
&SSHClient(IDC, me, you, CAlgs) = \\
&\quad send!me!you!IDC \rightarrow receive!you!me?IDS \rightarrow \\
&\quad \sqcap_{cookieC \in Cookies}\, send!me!you!(cookieC, CAlgs) \rightarrow \\
&\quad receive!you!me?(cookieS, SAlgs) \rightarrow \\
&\quad g := Negotiate(CAlgs, SAlgs, \text{`}g\text{'}) \in Parameter; \\
&\quad p := Negotiate(CAlgs, SAlgs, \text{`}p\text{'}) \in Parameter; \\
&\quad \sqcap_{x \in DHSecrets}\, send!me!you!EXP(g, x, p) \rightarrow \\
&\quad receive!you!me?(PubKeyS, DHPublicS, [\{H(finalHash)\}]_{PriKeyS}) \rightarrow \\
&\quad GO(EXP(DHPublicS, x, p), finalHash, PubKeyS, IDS)
\end{aligned}
$$

Figure 2. A possible abstract model of an SSH-TLP client.

client sends a nonce $cookieC$, followed by the client lists of supported algorithms $CAlgs$. The server responds sending a nonce $cookieS$, followed by the server lists of supported algorithms $SAlgs$. The client then computes the value of the Diffie Hellman (DH) parameters g and p with the $Negotiate(CAlgs, SAlgs, Param)$ function, which returns the requested negotiated algorithm parameter named $Param$, obtained from the supported client and server algorithms $CAlgs$ and $SAlgs$. Note that $\{CAlgs\} \cup \{SAlgs\} \subseteq Parameter$. Once g and p have been obtained, the client sends its DH public key $EXP(g, x, p)$, which is a message representing $g^x \bmod p$. This message is added to the datatype and the intruder knowledge derivation relation $\vdash$ is updated as well.

The following additional property of the $EXP(\cdot)$ function must also be modeled

$$EXP(EXP(g, y, p), x, p) = EXP(EXP(g, x, p), y, p) \tag{6}$$

in order to represent the identity of the keys built at client and server sides.

When the EXP function is used for the DH key exchange algorithm, x and y can be considered as DH private keys, $EXP(g, x, p)$ and $EXP(g, y, p)$ as DH public keys, and the expression in (6) as the DH shared key that can be obtained by each actor. Finally, g and p are the DH group parameters, that must be explicitly represented in the datatype, in order to express the exponentiation property. In the next step of the protocol, the client receives a message containing the opaque server public key $PubKeyS$, the opaque server DH public key $DHPublicS$ and the server signed final hash $[\{H(finalHash)\}]_{PriKeyS}$. Note that, as prescribed by the signature algorithm, the server signature of $finalHash$ is performed by encrypting, with the private key, the hashing of the message to be signed, rather than the message itself. The server will compute its DH public key as $DHPublicS = EXP(g, y, p)$, where y is the server's DH private key. However the client is modeled to receive an opaque $DHPublicS$ message, because the server DH public key is an opaque value from the client's point of view. Analogous reasoning holds for public and private server keys $PubKeyS$ and $PriKeyS$. The server $finalHash$ is the value upon which agreement is required, and contains all the relevant data of a protocol session, i.e.

$$
\begin{aligned}
finalHash = H(&IDC, IDS, (cookieC, CAlgs), (cookieS, SAlgs), PubKeyS, \\
&EXP(g, x, p), DHPublicS, EXP(DHPublicS, x, p))
\end{aligned}
$$

The $EXP(DHPublicS, x, p)$, that is used inside the final hash, is the DH shared key as computed by the client, that is the session main secret shared between the client and the

server. Finally, the $GO(\cdot)$ process is defined as

$$\begin{aligned} &GO(DHKey, finalHash, PubKeyS, IDS) = \\ &\quad (claimSecret.me.you.DHKey \rightarrow finished.me.you.finalHash) \\ &\quad \not< PubKeyS == TrustedKeyOf(IDS) \not> STOP \end{aligned}$$

where $P \not< b \not> Q$ means *if b then P else Q*. That is, if the received server public key $PubKeyS$ corresponds to the locally stored trusted key for the server identified by IDS, which is retrieved by the function $TrustedKeyOf(IDS)$, then the protocol run ends well, and all security properties can be claimed, namely the secrecy of the DH shared key $DHKey$, and the agreement on the server signed $finalHash$. Note that agreement on $finalHash$ implies agreement on all the data items on which it is computed. However, since the final hash is used later on with other data in order to establish a set of session keys, it is required that actors agree explicitly on $finalHash$, and not only on its contents.

Now that an abstract model of the client has been introduced, the refined model can be obtained by replacing the term $[\{H(finalHash)\}]_{PriKeyS}$ by

$$[\{(SignMode, SignPadding), H(SignHashAlg, finalHash\_enc)\}]_{PriKeyS}$$

and the term $finalHash$ by

$$\begin{aligned} finalHash\_enc = H(&FinalHashAlg, (IDC, IDS, (cookieC, CAlgs), (cookieS, SAlgs), \\ &PubKeyS, EXP(g, x, p), DHPublicS, EXP(DHPublicS, x, p))) \end{aligned}$$

Finally, the cryptographic algorithms and parameters denoted by the terms $FinalHashAlg$, $SignHashAlg$, $SignMode$ and $SignPadding$ are now explicitly represented and negotiated like the g and p parameters in the abstract model. In this refined model, each cryptographic function model includes cryptographic algorithms and parameters.

If secrecy can be verified on the abstract model, it can be implied on the refined one too, by putting the messages that are in the $Parameter$ set in the initial intruder knowledge when verifying secrecy on the abstract model, without even the need of writing down the refined model at all. In fact, assuming the intruder knows cryptographic algorithms and parameters is a reasonable assumption.

Agreement can be implied too on the refined model. In this case, it is required that, in the abstract model, the $finished$ action is defined as

$$finished.me.you.(finalHash, FinalHashAlg)$$

Note that this condition is needed even though the negotiated algorithm is already included in $CAlgs$ and $SAlgs$, because it is necessary to ensure that the specific negotiated algorithm is agreed, rather than the sets of algorithms from which it is selected.

5: Conclusions

The work presented in this paper is a useful step towards the verification of refined Dolev-Yao security protocol models. The main contribution is the formulation of a set of sufficient conditions under which the specification of which cryptographic algorithms and parameters are actually used can be safely abstracted away. It has been shown that, in

order to verify secrecy on the refined model, it is enough to verify this property on the corresponding abstract model, provided that the intruder knows the cryptographic algorithms and parameters, which is a reasonable assumption. For authentication, the abstract model can be verified, provided verification also checks agreement on the cryptographic algorithms and parameters used to build the protocol messages upon which agreement is required.

The model of cryptographic algorithms and parameters that has been developed in this paper is general enough to take into account run-time algorithms and parameters negotiation, and separate, independent modelling of algorithms and parameters for key construction, encryption and hashing operations.

Although these results are not so surprising, they have been formally stated for the first time in this paper, and they find application in improving the development of formally verified implementation code of security protocols, both using the code generation approach or the model extraction approach.

With code generation, the developer only needs to write and verify the abstract protocol model, and the code generation engine can take care of using the right cryptographic algorithms and parameters. When adopting a model extraction approach, it is possible to disregard cryptographic algorithms and parameters, and extract a simpler model.

Some issues on the topics presented in this paper are still open for future work. In particular, it can be interesting to explore under which conditions the fault preserving simplifying transformations are safe for other security trace properties. Another interesting further work is to consider verification with computational models instead of verification with Dolev-Yao models, which is limited because of its abstract view of cryptography.

References

[1] Karthikeyan Bhargavan, Cedric Fournet, Andrew D. Gordon, and Stephen Tse. Verified interoperable implementations of security protocols. In *Computer Security Foundations Workshop*, pages 139–152. IEEE Press, 2006.

[2] Danny Dolev and Andrew Chi-Chih Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–207, 1983.

[3] Jean Goubault-Larrecq and Fabrice Parrennes. Cryptographic protocol analysis on real C code. In *Verification, Model Checking, and Abstract Interpretation, LNCS 3385*, pages 363–379. Springer, 2005.

[4] Mei Lin Hui and Gavin Lowe. Fault-preserving simplifying transformations for security protocols. *Journal of Computer Security*, 9(1/2):3–46, 2001.

[5] Alfredo Pironti and Riccardo Sisto. Reasoning about some security protocol implementation details, revision 4. Technical Report, Politecnico di Torino, January 2008. Available at: http://staff.polito.it/riccardo.sisto/reports/encoding_4.ps.

[6] Davide Pozza, Riccardo Sisto, and Luca Durante. Spi2java: Automatic cryptographic protocol java code generation from spi calculus. In *International Conference on Advanced Information Networking and Applications*, pages 400–405. IEEE Press, 2004.

[7] A. W. Roscoe. *The Theory and Practice of Concurrency*. Prentice Hall, 1997.

[8] Benjamin Tobler and Andrew Hutchison. Generating network security protocol implementations from formal specifications. In *Certification and Security in Inter-Organizational E-Services*, pages 33–54. Springer, 2004.

[9] T. Ylonen and C. Lonvick. The Secure Shell (SSH) Protocol Architecture. RFC 4251 (Proposed Standard), January 2006.

[10] T. Ylonen and C. Lonvick. The Secure Shell (SSH) Transport Layer Protocol. RFC 4253 (Proposed Standard), January 2006.