A Formal Approach for Automatic Verification of Imperfect Cryptographic Protocols

Long Nguyen Hoang
University of Tartu, Institute of Computer Science

December 5, 2007

1 Introduction

In simplest form, security protocols comprise messages exchanged between agents to achieve security goals such as confidentiality and integrity of data, or authentication of the identity. Despite that simple fact, designing security protocols has been considered critical task since the protocols should work in the presence of powerful adversary over the network. Analyzing security protocols is non-trivial as it exploits undiscovered security holes. Technically, one security protocol can be modeled and analyzed in two perspectives: the formal model and the computational model. The former approach assumes perfect cryptography: having an encrypted message $\{M\}_K$, there is no way to recover original message M without knowing secret key K; also it is infeasible to retrieve knowledge of K from $\{M\}_K$. Over the years, based on this assumption, effective methods and automated tools have been developed and studied to help us to analyze the design of protocols. However, while assumption of perfect cryptography strengthens the process of security protocol analyzing, it also restricts us to a limited, fixed security model. There are situations where applications of one security protocol are not perfectly secure as they are proved because of the gap between the formal representation of encryption and its concrete implementation. Computational model, on the other hand, defines security properties in terms of the probability and computational complexity of successful attacks. Hence perfect encryption assumption and adversary power have been relaxed, respectively. In computational view, good security protocols are those in which adversaries cannot cause harm too often and efficiently enough.

Each approach above has its own advantages and drawbacks. These models, unfortunately, are quite distinct as they are developed by two separated communities. Such efforts have been proposed to take benefit from both models

[DDMR07, AR00, Cre07] but none of them solves the problem in completed way. In this paper, we introduce a completed probabilistic framework for formal analysis of security protocols including consideration of imperfect cryptography and probabilistic intruder model. As a result, we are able to reason about security protocols as in formal view and aware of the power of adversary to break encrypted messages as in computational view. In the other words, security protocols can be automatically verified with better accuracy because they have been verified in both formal model and computational model.

The rest of this paper is structured as follows. In Section 2 we envision our framework by extending spi-calculus to take into account the "imperfect" cryptography and considering the new role of the adversary in our model. Section 3 provides proof of the correctness of our contribution. After that, Section 4 raises discussion about how our framework can be applied. Finally, the conclusion is formed in Section 5.

2 A probabilistic framework for formal analysis of security protocols

The main drawback of security protocol analysis in formal model is that it restricts us to a fixed model of adversary, ignoring the possibility of adversary doing some statistic attacks such as guessing or chosen-ciphertext attacks. Strictly speaking, the adversary studied in this paper has the power as that in the Dolev-Yao model[DY81] plus the ability to guess key or to obtain plaintext from ciphertext with a certain probability. In order to allow an adversary to exploit security properties from encrypted ciphertext in formal security model, there are two issues we need to consider: a semantic for reasoning about imperfect cryptography and a probabilistic adversary model in the context of that semantic.

2.1 Semantic for imperfect cryptography analysis

We first need a notion of probabilistic pattern to express the probability of retrieving useful information from cryptography expression. Semantics to abandon perfect cryptography assumption is introduced in [TAG03]. Set **pPat** of probabilistic patterns is defined by the grammar below:

$P_{\cdot p},\ Q_{\cdot p} ::=$	probabilistic patterns
$K_{\cdot p}$	key (for K $\in$ **Keys**)
$m_{\cdot p}$	string (for m $\in$ **String**)
$(P_{\cdot p},\ Q_{\cdot p})_{\cdot p}$	pair
$p \in [0, 1]$	

Given an adversary A and its initial knowledge G, the probability of A obtaining m from $\{m\}_K$ without knowing K is calculated by

$$Pr[m \leftarrow A(\{m\}_K,\ G)] \leqslant p_{dec}(\{m\}_K,\ G) \text{ for all } A$$

The value of p_{dec} depends on knowledge G and the strategy being used by the cryptanalysis. In such a case, if $G \rightarrow K$ then $p_{dec}(\{m\}_K, G) = 1$. On the other hand, if ideal encryption scheme has been reached then the power of the adversary is restricted to that in Dolev-Yao mode (i.e. p_{dec} is negligible). However, using just function p_{dec} is not enough to specify the probability of decrypting a ciphered text; we need more functions to evaluate it.

Let M be an expression, G be initial knowledge of the adversary, $T \subseteq Keys$ be set of keys that can be obtained from the expression M, and $p \in [0, 1]$ be probability of breaking all the keys contained in T by applying a particular strategy. We have 4 parameters $pKeys_M^G$, $pGuess_M^G(T)$, $pMax_M^G$ and pP_M^G such that $pKeys_M^G$ contains set of keys that are possible to be obtained from expression M, $pGuess_M^G(T)$ is the maximum probability of breaking all keys in T, $pMax_M^G$ describes the maximum probability of getting information about plaintext contained in the expression M, and finally pP_M^G is used to turn an expression M into a probabilistic pattern.

Initially, we assume that the adversary can reduce the key space by applying some techniques; otherwise he can do nothing but randomly guess or try brute-force attack which is trivial. The set $pKeys_M^G$ is generated by first applying *initKeys* to get set of initial keys, then recursively adding sets of new key that can be obtained somehow from $pKeys_M^G$ by using *addKeys* function. Formally, $pKeys_M^G$ is specified as follows:

$$pKeys_M^G = \{initKeys((M, G))\} \quad \text{then} \quad addKeys((M,\ G),\ 1)$$

addKeys(H,p) ::=
$\forall \{N\}_K$: $(H \mapsto \{N\}_K \wedge H \not\mapsto K)$ do begin
p' $= p.p_{dec}(\{N\}_K,\ H)$
L $= (H,\ K)$
T $= \{\ K \in Keys \mid L \mapsto K\ \}$
$pKeys_M^G$ $= pKeys_M^G \cup \{T._{p'}\}$
$addKeys(L,\ p')$
end

Secondly, $pGuess_M^G(T)$ is calculated. It is noted that T is a subset of $pKey_M^G$.

$$pGuess_M^G(T) = max\{p | J._p \in pKeys_M^G \wedge T \subseteq J\}$$

Thirdly, $pMax_M^G$ expresses the maximum probability of guessing all the keys used in M.

$$pMax_M^G = max\{p | J._p \in pKeys_M^G \wedge allKeys(M) \subseteq J\}$$
or $pMax_M^G = pGuess_M^G(allKeys(M))$, alternatively,

Finally, pP is a function $pP_M^G : Exp \times D_{pGuess_M^G} \rightarrow pPat$ defined inductively

$$\begin{aligned}
pP_M^G(K, T) &= K_{pGuess_M^G(T)} && (K \in \mathbf{Keys}) \\
pP_M^G(m, T) &= m_{pGuess_M^G(T)} && (m \in \mathbf{String}) \\
pP_M^G((N_1, N_2), T) &= (pP_M^G(N_1, T), pP_M^G(N_2, T))._{pGuess_M^G(T)} \\
pP_M^G(\{N\}_K, T) &= pP_M^G(N, T') && (T' = T \cup \{K\})
\end{aligned}$$

Now we have enough semantics to approximate indistinguishability of security expressions. The reason we have to approximate because it is rarely to find blocks with exactly the same probability of being attacked. Probabilistic indistinguishability is a very important security notion because one adversary can do nothing but randomly guessing with indistinguishable expressions. We say that expressions M and N are *probabilistically equivalent* if and only if

$$M \approx_\varepsilon N \leftrightarrow pP_M \sim_\varepsilon pP_N \wedge |pMax_M - pMax_N| \leqslant \varepsilon$$

where two probabilistic patterns are defined to be indistinguishable as

$$\begin{aligned}
&P._p \sim_\varepsilon Q._{p'} && \textit{iff } p,p' \leqslant \varepsilon && P._p, Q._{p'} \in \mathbf{pPat} \\
&K._p \sim_\varepsilon K._{p'} && \textit{iff } |p - p'| \leqslant \varepsilon && K \in \mathbf{Keys} \\
&m._p \sim_\varepsilon m._{p'} && \textit{iff } |p - p'| \leqslant \varepsilon && m \in \mathbf{String} \\
&(P._{p_1}, Q._{p_2})._{p_3} \sim_\varepsilon (P'._{p'_1}, Q'._{p'_2})._{p'_3} && \textit{iff } |p_3 - p'_3| \leqslant \varepsilon \wedge \\
& && P._{p_1} \sim_\varepsilon P'._{p'_1} \wedge Q._{p_2} \sim_\varepsilon Q'._{p'_2} \\
& && P._{p_1},\ P'._{p'_1},\ Q._{p_2},\ Q'._{p'_2} \in \mathbf{pPat}
\end{aligned}$$

The next thing we consider is a process calculus to formally express security protocols and reason on it. In order to make use of the semantic above, we propose a new process calculus extended from spi calculus [AG97]. We call that *probabilistic-spi calculus*. The reason we choose to inherit from spi-calculus is because it provides a formal semantics for analyzing security protocols. Moreover, traditional attacks can be detected by reasoning on spi calculus. The set of terms is defined by the grammar:

$L,\ M,\ N ::=$	terms
$\ldots$	as in spi-calculus
$\{M\}_{K.p}$	probabilistic encryption

where $\{M\}_{K.p}$ means that the encrypted message $\{M\}_K$ can be decrypted by an adversary without knowing key K with probability p. The set of process is defined by the grammar:

$P,\ Q ::=$	processes
$\ldots$	as in spi-calculus

We observe that formal semantics of spi calculus still holds in this extension if we leave out the probability (i.e. $\approx_0$). It means that formal analysis in spi calculus again can be applied in probabilistic-spi calculus. With $\varepsilon \in (0;\ 1]$, such expressions like P(M) and P(M′) may become distinguishable because an observer now has the probability to discover key K and hence can tell whether M or M′ is sent. Therefore, it is important to prove that our modification to spi-calculus doesn't expose any new attacks but those we have expected. Proof of correctness of probabilistic-spi calculus is provided in Section 3.

2.2 Probabilistic intruder model

In our model, the intruder is allowed to retrieve certain information from the messages he has captured with some probability of success. Although the probability is low, gathering enough information may significantly increase intruder's chance thus open new ways for attacks. In other point of view, the power of the adversary is extended and formalized from that in Dolev-Yao intruder model[DY81]. Such traditional attacks in formal model can be modelled using our calculus. In this part we focus on how attacks like those in computational model can be modelled and verified using our calculus. To sum up, we assume the adversary has ability to:

- recognize what protocol is being used
- capture and record any messages sent over the network to its memory for later examination.
- randomly guessing key or applying knowledge such as statistics analysis to decrypt the encrypted message
- somehow reduce the key space hence gain better chance to attack successfully

Here we need to verify whether the protocol stays secure or not when the intruder can learn a certain message with given probability p. Verification process starts with expressing the protocols' states as a *probabilistic computation tree* whose root is the initial state while nodes in tree are states of protocol and connected edge indicates transition from one state to another. Each edge is labeled with probability of the adversary to attack the single message studied in Section 2.1. For transitions which the adversary doesn't care, the probability are marked 1. Figure 1 depicts an example of probabilistic computation tree.

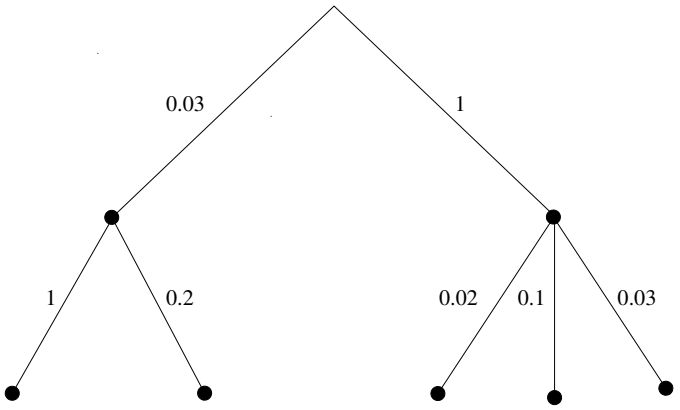

Figure 1: A probabilistic computation tree.

Once the tree is completely built, we can compute the probabilities associated with the branches of the tree by traversing backwards the attack trace to the root then multiplying all the probability label; as a result we can to tell with which probability the attack is possible in the end. Given a protocol P = (S,R,AR) and probability p where S, R and AR are respectively set of states, set of transition rules and set of rules for identifying an attack state. The recursive algorithm *qModelCheck* for model checking is defined as follows:

qModelCheck(P)
Input: Protocol P = *(S,R,AR)* and probability p
Output: 1 if the protocol is secure, 0 otherwise
1. If ($ispAttack_{AR}(S) \wedge pAttack_{AR}(S) \geq p$) then Return 0;
2. Compute the set $pApplicable_{lhs}(S)$ for each $r \in R$ with $r = lhs \Rightarrow rhs$;
3. Compute $pSucc_R(S)$;
4. For each $S' \in pSuccR(S)$;
5. B=qModelCheck(S',R,AR);
6. If(B==0) then Return 0;
7. Return 1.

The attack predicate $ispAttack_{AR}(S)$ is true if and only if the attack rule AR can be applied to the state S. *pApplicable* is that function that maps a state S and the left-hand side *lhs* of the attack rule r to the *ground* state whose set of variables is *zero*. *pSucc* returns the corresponding set of successor states that can be attacked. And finally, *pAttack* is the probability of such an attack computed by multiplying all probabilities in set of states S. We refer reader to [AMRV06] for more detail. It is noted that the computation of *pApplicable* assumes there exist pre-computed values of probability *prule* in the beginning. Fortunately, we realize that the pre-computed *prule* matches to $pGuess_M^G$ in Section 2.1.

Our model can be related to computational model for further security analysis. In order to do that, expressions are mapped into computational model in which they are seen as random bit-strings with the same length as the size of their domains in formal model.

3 Proof of correctness

Theorem: *When the probability of the intruder is zero (i.e. ideal cryptography), probabilistic-spi calculus is restricted to spi-calculus.* This also means semantics and security properties of spi calculus still hold in probabilistic-spi calculus. Therefore, safe/unsafe in formal model (spi implies safe/unsafe in our model.

Proof : Using the syntax in [TAG03], we have:

$$M \approx_0 N \Longleftrightarrow M \approx N$$
$$M \approx N \Longleftrightarrow pP_M = pP_N \wedge pMax_M = pMax_N$$
$$pP_M = pP_N \Longleftrightarrow pat(M) = pat(N)$$

Therefore, we have:

$$M \approx_0 N \Longrightarrow M \cong N$$

We have proved that, in zero-probability, probabilistic equivalence is restricted to formal indistinguishability. It means proof of correctness of spi-calculus also holds in our model. This allows us to reason on formal model in case we assume perfect/ideal cryptography.

Theorem: *Safe/unsafe in probabilistic-spi calculus implies safe/unsafe in computational model.*

Proof: A process, in computational model, is considerably safe when function Adv(n) is *negligible* for all probabilistic polynomial-time algorithm $A^{(.),(.)}$ in n, otherwise it is unsafe.

$$Adv(n) = Pr[k, k' \leftarrow G(1^n): A^{\varepsilon(1^n,k,.),\varepsilon(1^n,k',.)}(1^n) = 1]$$
$$- Pr[k \leftarrow G(1^n): A^{\varepsilon(1^n,k,0),\varepsilon(1^n,k,0)}(1^n) = 1]$$

Consider an adversary that can obtain useful information from encrypted ciphertext with a non-negligible chance of success. That adversary can be expressed as a probabilistic Turing machine in sub-exponential polynomial time. We prove that, if there exists an attack on a protocol in the probabilistic-spi calculus then the equivalent system in computational model can be broken by a polynomial time Turing machine. It's obvious to see if that Turing machine can attack the protocol in our model, the equivalent protocol in computational view can be attacked with the same Turing machine.

We now prove the "*safe*" implication. Let P be a process that has no sub-exponential attack. Assume the contradiction that there exists an probabilistic Turing machine A such that Adv(n) is non-negligible in the probabilistic-spi calculus while no such attacker exists for spi-calculus. It means the attacker can defeat the Turing encoding of the probabilistic-spi calculus process with non-negligible chance of success, but cannot defeat the Turing encoding of the spi-calculus process. We consider a process P. As the cost of the attack is subcase-exponential we know that the attacking process has a finite number of sub-exponential names. We can write a simulator of P in spi calculus so that they have the same names. Because A can break P, we can construct a similar machine A2 that breaks the simulator. However, since the simulator is considered in spi calculus, there must exists a spi-calculus attacker that can defeat it, which fails the contradiction.

4 Discussion

It now suffices to put our framework into application. The most advantage is that one protocol can be modelled like that in formal model while it can be analyzed like that in computational model. In one hand, attacks that could be detectable in the Dolev-Yao model can be detected too in our model by ignoring the probability. Moreover, because our model extends from spi-calculus, well-known automatic verification tools such as [EHHN99, DSD04, Bla01] may be applied. In the other hand, we have proved in Section 3 that reasoning on our framework gives a similar result as we do in computational model and our model doesn't expose any unexpected attacks.

Let us consider an abstract and simplified version of Wide Mouthed Frog protocol. Informally, the protocol can be expressed as follows:

$$\begin{aligned}&\text{Message 1 } A \rightarrow S : \{K_{AB}\}_{K_{AS}} \text{ on } c_{AS}\\ &\text{Message 2 } S \rightarrow B : \{K_{AB}\}_{K_{SB}} \text{ on } c_{SB}\\ &\text{Message 3 } A \rightarrow B : \{M\}_{K_{AB}} \text{ on } c_{AB}\end{aligned}$$

Following our semantics, we rewrite using probabilistic-spi calculus for further consideration:

$$A(M) \triangleq (\nu K_{AB})(\overline{c_{AS}}\langle\{K_{AB}\}_{K_{AS}\cdot p_{dec_1}}\rangle).(\overline{c_{AB}}\langle\{M\}_{K_{AB}\cdot p_{dec_2}}\rangle)$$

$$S \triangleq c_{AS}(x).case\ (x)\ of\ \{y\}_{K_{AS}}\ in\ \overline{c_{SB}}\langle\{y\}_{K_{SB}\cdot p_{dec_3}}\rangle$$
$$B \triangleq c_{SB}(x).case\ (x)\ of\ \{y\}_{K_{SB}}\ in$$
$$c_{AB}(z).case(z)\ of\ \{w\}_y\ in\ F(w)$$
$$Inst(M) \triangleq (\nu K_{AS})(\nu K_{SB})(A(M) \mid S \mid B)$$

where *F(w)* is function representing the rest of the behaviour of B upon receiving message w.

This protocol is obviously vulnerable to attacks caused by adversaries who intercept a message and replace it. Here we only appraise those who listen passively and try to forge the message without knowing the shared keys. The protocol can be analyzed using methods in Section 2. It is noted that the probability is higher than probability of randomly guessing otherwise this consideration is negligible. For example, if $\{K_{AB}\}_{K_{AS}}$ and $\{K_{AB}\}_{K_{SB}}$ are encrypted by the same mechanism that is vulnerable to plaintext attack then the adversary can gain significant knowledge and greatly reduce the key space. Also, the attacker has better chance if he attacks channel AS and SB than trying to attack directly channel AB.

5 Conclusion

In this paper we have introduced a "hybrid" framework that has strength of both formal model and computational model. We have extended the power of the adversary in formal model so that it is allowed to carry such attacks in computational model. Compared to other methods [DDMR07, LMMS98, AR00], our model allows reasoning directly on process calculus rather than studying the soundness of formal encryption. [DDMR07] can be applied in both formal model and computational model but it proposes an intermediate language, which is much more complex than our approach. [AR00] does the job in the opposite way: they turn a protocol from formal model to one in computational model then reason on that translation. The main reason we decide to work with formal method is because we want to make use of existing formal automatic verification tools. Security holes in logic can be exploited by those tools while statistic attacks can be detected using method mentioned in Section 2. Another interesting use case of our framework is that a security protocol our model can be easily transformed to relevant one in computational model for further consideration. In future, implementation of automatic verification tool for our model seems worth trying. Other model checking methods such as [BMV04] may be studied to see if they can apply in our framework as well.

References

[AG97] Martin Abadi and Andrew D. Gordon. A calculus for cryptographic protocols - the spi calculus. pages 36–47, 1997.

[AMRV06] Pedro Adao, Paulo Mateus, Tiago Reis, and Luca Vigano. Towards a quantitative analysis of security protocols. pages 164–167, 2006.

[AR00] Martin Abadi and Phillip Rogaway. Reconciling two views of cryptography(the computational soundness of formal encryption). pages 3–22, 2000.

[Bla01] B. Blanchet. An efficient cryptographic protocol verifier based on prolog rules. pages 82–96, 2001.

[BMV04] D. Bain, S. Modersheim, and L. Vigano. Ofmc: A symbolic model checker for security protocols. 2004.

[Cre07] C. Cremers. On the Protocol Composition Logic PCL. *ArXiv e-prints*, 709, September 2007.

[DDMR07] Anupam Datta, Ante Derek, John C. Mitchell, and Arnab Roy. Protocol composition logic (pcl). pages 311–358, 2007.

[DSD04] D. Pozza D, R. Sisto, and L. Durante. Spi2java: automatic cryptographic protocol java code generation from spi calculus. pages 400–405, 2004.

[DY81] D. Dolev and A. C. Yao. On the security of public key protocols. pages 350–357, 1981.

[EHHN99] A. Elkjar, M. Hohle, H. Huttel, and K. Nielsen. Towards automatic bisimilarity checking in the spi calculus. 1999.

[LMMS98] P. Lincoln, J. Mitchell, M. Mitchell, and A. Scedrov. A probabilistic poly-time framework for protocol analysis. April 3, 1998.

[TAG03] Angelo Troina, Alessandro Aldini, and Roberto Gorrieri. A probabilistic formulation of imperfect cryptography. June 2003.