**Abstract**

We propose to design, develop and experiment a new approach of formal security proofs.

One of the main issues in formal security proofs is to design an attacker model. Such a model is always disputable; even when a proof is completed in some model, one can find an attack in another model.

This project aims at bypassing the problem, roughly by specifying what an attacker *cannot* do, instead of specifying what he/she can do.

1 The context

1.1 Software security

The development of distributed applications that send and receive information from other sources is both an opportunity and a threat. Let me give some examples.

Internet banking and, more generally Internet transactions, increases our comfort, avoiding a move (and possibly a queue) at the bank/merchant’s location. Furthermore, the services are available at any time and from any place. It is therefore an opportunity, but it is also a threat since confidential information, such as credit card numbers or any other credentials, is sent over an insecure communication channel.

Mobile phones offer geo-localisation, which is very convenient for finding our way. It is also a threat since this information could be eavesdropped, while we wish to keep it private. This observation applies to many on-line services that are very convenient, but leak information about ourselves. Big Brother is not far.
In the medical area, it will be very convenient to wear devices that send measures such as our blood pressure, our temperature... and receive instructions from the medical center on how the device should react. However, we can imagine that an attacker could send wrong information. In 2013, the US vice-president Dick Cheney feared a terrorist attack on his pacemaker.

This observation also applies to many other embedded devices. In 2015, we were not surprised to learn that it is possible to mount remote attacks on a vehicle, taking the control of the car (http://argus-sec.com/car-hacking/).

Not only terrorists are a threat. As the PRISM scandal shows, also those whose intentions are noble may get some information that we want to keep confidential.

There are numerous other examples. There will be more and more in the future. We wish to take advantage of these new technologies. We also wish to limit the threats as much as possible. To achieve this goal and get the best without the risks, the programs aim at achieving security goals. They use security primitives such as encryption, digital signatures, zero-knowledge proofs ... Can we trust them?

1.2 Safety vs security. The role of formal methods

The issue of “hunting the bugs” is not new. Several methods and tools have been designed in the past 50 years. Typically, embedded software (on cars, aircrafts, spaceships) attracted some attention. This yield verification algorithms, proof systems, testing methods... that have proved their usefulness and have significantly increased the safety of critical applications: the programs are more often implementing their expected functionality.

Security is different from safety. In the case of security properties, we must show that some property is satisfied, in the presence of an arbitrary adversary. At first sight this looks similar to the verification of a controller, whose goal is to bring back the system to a safe state, whatever the actions of the environment are. There is however a major difference. For classical safety properties, the actions of the controller depend on its observations. In the case of security, the (honest) programs may not even notice that an attack is going on: often there are no observations available.

Therefore the classical model-checking techniques that have been developed during the last decades cannot be (at least not directly) used for security.

Similarly, other techniques such as statistical testing are not relevant,
because whatever distribution of events is used for testing, the attacker need not comply with such a distribution. More generally, we can expect an attacker to rely on any security breach whatsoever. That is why formal methods play an important role in increasing our confidence in the security of programs.

In the remainder of the project, we will focus on security protocols (the distributed programs, as in the above examples). The same ideas might be used for other programs/circuits aiming at some security goal. This is not discussed in this project, but might be the subject of future work.

1.3 What is an attacker?

We have explained why formal methods play an important role in computer security. However, a formal analysis of a security application/protocol requires a formal definition of the attacker. This is a central question and a starting point of our project. We first investigate the various attacker models that have been considered so far.

The Dolev-Yao attacker  In their seminal work D. Dolev and A. Yao [21] define a formal model, in which the attacker is a device that sits in the middle of every communication. The attacker eavesdrops on all messages, builds new messages according to a fixed set of rules and sends messages (impersonating the sender). The scope of their framework is very limited (only public-key encryption, a fixed set of pairs sender/receiver, a “ping-pong” control structure,...) but since then, many extensions have been designed along the same lines. The protocols are now specified in a process algebra that embeds security primitives, such as the applied pi calculus [2]. Several automatic verification tools have been designed, relying on what is now known as “the Dolev-Yao attacker”. Among the most recent ones, let me cite ProVerif [18] or Scyther [20], but there are many more.

Following Dolev and Yao, all these works see messages as terms constructed on black-box cryptographic primitives, whose properties are specified equationally. The attacker can only use such primitives to build or analyse the messages.

While this approach has been proved to be successful in finding attacks (see for instance [1]), proofs of protocols in these models should be taken with care. Indeed, the attacker model is a bit rough; nothing prevents an attacker from using any other computation mechanism than the fixed set of primitives that are given to him/her in this model. For instance there are protocols that have been proved secure in the Dolev-Yao model and for
which attacks were found later. This is the case for instance of the Needham-
Schroeder-Lowe (NSL) protocol [33] and Bull’s authentication protocol [30].

The latter example, as well as others, were a motivation for introducing
equational properties of the security primitives, or additional capabilities of
the attacker. Several tools such as ProVerif [18] or Tamarin [26] implement
some restricted properties of the primitives.

The other example (the attack on the NSL protocol) is more difficult.
It relies mainly on the malleability of an encryption scheme: the attack can
be mounted if it is possible to build a ciphertext from another ciphertext,
without holding the decryption key. This is possible for instance with the
El Gamal encryption scheme [22]. If we wish to capture such an attack,
we need to give the attacker such a capability. What should exactly be the
extra capabilities of the attacker is however not clear. It depends on the
actual encryption scheme and on the implementation of other primitives.

**The computational attacker**  A more accurate attacker model has been
adopted by the cryptographers, following Goldwasser and Micali [23]. As
in the Dolev-Yao case, the attacker may eavesdrop on messages and build
fake messages. However, he/she is not any more limited by a fixed set of
possible operations. He/she may now perform any computation that can
be completed in probabilistic polynomial time (in the worst case). The
input of such an attacker is a security parameter (roughly, the length of the
encryption keys).

Now, there is always a chance that an attacker will succeed, since he/she
may simply guess the key correctly. In this context, a protocol is secure
if the success probability of an attacker is bounded by a function of the
security parameter, which decreases quickly when the security parameter
grows; security is an asymptotic property.

This model of attacker is currently the most accurate one for which
formal proofs and formal tools are available. For instance CryptoVerif
[13] is a prover that automatically reduces cryptographic games to simpler
games. In the initial game, the attacker plays against the protocol and in
the final game an attacker plays against a presumably hard problem (such
as discrete logarithm, or IND-CPA if the encryption scheme is assumed to
satisfy such a property). The probability of success of an attacker on a
given game is computed as a function of the probability of success of the
attacker in the previous game. Then, the protocol is secure if the probability
of success of the attacker in the initial game is not significantly larger than
the probability of success of the attacker on the presumably hard problem.
CryptoVerif has had a number of successes in proving protocols in this setting. Though the library of games and strategies is continuously increasing, there are still many situations that require the user to add a new game or to interact with the tool in order to guide the reduction strategy. In some cases, CryptoVerif simply fails to prove the protocol.

EasyCrypt [9] shares several features with CryptoVerif: it also works by successive reductions, however at a lower level, as it deals with individual commands. This allows for more flexibility and, in principle, a broader scope of applications. It is however a proof assistant, which requires interactions with the user. So far, it was mostly successful in proving/attacking complex cryptographic primitives (see for instance [10]), not yet full protocols.

In both cases, there is no full automation: the default strategy is likely to fail on many inputs. Another issue is the composability of proofs, which is necessary for relatively large protocols. Most importantly, proof attempts may fail. Does it mean that we have not put sufficient effort in completing the proof? Or does it mean that there is an attack?

On the other hand, these tools provide the user with quantitative information on the probability of success of an attacker (which we do not expect to obtain in our project).

Computational soundness Another line of research, which was initiated by M. Abadi and P. Rogaway [3], consists in trying to get the best of the previous two models (the Dolev-Yao one and the computational one). The idea is to prove a “full abstraction” result for a Dolev-Yao style attacker. Such results are now known as “computational soundness results”. They list the computational assumptions, under which a Dolev-Yao attacker is equally (up to negligible probability) likely to mount an attack than the computational attacker.

Abadi and Rogaway [3] consider a passive attacker, symmetric encryption and equivalence properties. Further papers, for instance [6, 4, 19, 15] but there are many more, extend this idea to active attacks, other primitives and/or other security properties.

Those works provide nice hints on which properties of the primitives are relevant. As shown in [16], they require a lot of assumptions, some of which are not realistic: no key cycle is ever created (this is undecidable), the keys are honestly generated (meaning that they always come with a certificate), there is a polynomial parsing algorithm from bitstrings to terms (this does not exist for some primitives),.... Furthermore, the computational soundness
proofs tend to be extremely complicated, hence difficult to check.

**Modularity**  As the attacker model becomes more accurate, the proofs become more complex. When a protocol combines several subprotocols or uses several primitives, it is necessary to decompose the proofs. This was the main motivation for the introduction of the “universal composability” property [14]. This property is however difficult to prove (and actually to define precisely, as there were several successive definitions). It has also some limitations (see for instance [25]), related to the “commitment problem”: there is no UC proof for primitives that first commit on some hidden value and then reveal that value.

Up to now, there is no formal tool that would complete or check universal composability proofs.

**More powerful attackers**  One could also wonder whether the computational attacker, as defined above, is the best possible definition. There were several criticisms, mainly on the asymptotic nature of the security guarantees in this model [24]. But there are other questions. For instance, why is it a machine running in worst-case probabilistic polynomial time? why not (for instance) in average probabilistic polynomial time? Or subexponential probabilistic polynomial time? As far as I understand, the reason is to consider a class of machines, which is closed by composition; the motivations are not the accuracy of the model.

Anyway, we may realistically imagine more powerful attackers. For instance attackers that have some other channels through which they may get side information. It could be the observation of time or power consumption in the execution of a program, for instance. The attacker could also corrupt some package updates, which replace cryptographic primitives by fake implementations, yet undetectable by the user (see for instance [12]).

**Conclusion**  It seems that we are faced with a dilemma: the more accurate the attacker model is, the more difficult the security proofs are. Already for the computational attacker, it is very difficult to complete security proofs automatically.

Is such an increasing complexity of the proofs unavoidable when the attackers becomes more powerful? We will show that the answer is no. There is a way out.
2 Our approach

2.1 A universal attacker?

We propose in [7, 8] a new approach, which overcomes some of the above mentioned shortcomings. This approach is also described in G. Scerri’s thesis [31].

The main idea is to consider an attacker that can a priori perform any computation, unless it breaks some assumptions. In other words, instead of specifying the attacker’s capabilities, as the Dolev-Yao and computational model do, we specify what an attacker cannot do; we consider the most powerful attacker that does not break some (first-order) axioms.

Given such axioms $A$, a security property $\phi$ and the conditions $\theta$ that gather the tests performed along a protocol execution, the security of the protocol reduces to the inconsistency of $A \cup \{\neg \phi, \theta\}$. Conversely, the consistency of such a set of formulas implies the existence of a model, which witnesses an attack.

Example

Let us give a very simple example. The precise definitions are missing and the example might be difficult to follow in details. The goal is to give a flavor of the method.

We consider a logic that includes a variadic predicate $\triangledown$. The intended meaning of $T \triangledown t$ is that an attacker may compute $t$ from $T$. This can be interpreted in the computational setting, using a Kripke semantics, in which the worlds are the non-negligible sets of samplings; $T \triangledown t$ is valid in the computational semantics, if, for any non-negligible $S$, there is a non-negligible $S' \subseteq S$ and a probabilistic polynomial time Turing machine $A$, such that for the samplings $\tau \in S'$, given the interpretation of $T$ w.r.t. $\tau$, $A$ computes the interpretation of $t$ w.r.t. $\tau$.

We consider two axiom schemes:

\begin{align*}
\text{fresh}(X, n) : & \quad X \not\ni n \\
\text{usable}(X \cup x, k), \text{fresh}(X \cup x, r) : & \quad X, \text{enc}(x, k, r) \triangledown n \rightarrow X \triangledown n
\end{align*}

which states (roughly) that, if a name $n$ does not occur in a set $X$, then the attacker cannot compute $n$. In the computational semantics, this statement is valid, since, without any information on $n$, the attacker can only guess $n$ with a negligible probability.

\begin{align*}
\text{usable}(X \cup x, k), \text{fresh}(X \cup x, r) : & \quad X, \text{enc}(x, k, r) \triangledown n \rightarrow X \triangledown n
\end{align*}

which states that an encryption with a usable key $k$ does not bring any significant additional information for computing $n$; if $n$ can be computed with
the ciphertext, then it can be computed without it. The definition of usable depends on the assumptions on the encryption scheme. The simplest case is to assume that the encryption key $k$ does not appear as a plaintext in $X, x$ (only as an encryption key). In the computational model, such an axiom is valid for IND-CPA encryption schemes that are which-key concealing, as shown in [31].

Now, if a protocol consists in sending out a single message $\text{enc}(n, k, r)$ and we wish to prove that the random number $n$ is secret, it is sufficient to prove that the axioms, together with $\text{enc}(n, k, r) \triangleright n$ (the negation of the security property) are inconsistent. This is the case since, for an empty set $X$, the second axiom yields $\emptyset \triangleright n$, which contradicts the first axiom.

As a consequence, this trivial protocol is secure, for any attacker and cryptographic primitives that satisfy the two axioms. This is in particular the case for a computational attacker, when the encryption scheme is IND-CPA and which key concealing.

This approach is very appealing, as we could, for instance, complete proofs in the computational model, while staying within first-order logic. We do not need any of the hypotheses of computational soundness proofs and still avoid the machine reductions and probabilities computations. There are still a number of questions:

**Axioms** What are the axioms? How to design them? How can we show their relevance?

**Proof search** A priori, the consistency of first-order formulas is undecidable. Do we really gain something?

**Case studies** Is the method of any practical use? Can we discover new attacks?

**Modularity** Can we decompose the proofs of complex protocols into smaller pieces? Can we re-use some proofs?

**Security properties** Which security properties can be specified?

Some of these questions have been (at least partly) answered in the thesis of Guillaume Scerri (defended on January 2015), some others are part of this project.
2.2 G. Scerri’s thesis

**Axioms** In G. Scerri’s thesis, a number of axioms are proposed, which reflect some classical properties of encryption schemes, such as IND-CPA [23], or integrity [11]. They are proved to be valid in a computational model. Therefore, the inconsistency of the set of formulas implies computational security.

**Proof search** In general, the consistency of a set of first-order formulas is undecidable. However, in the case of the above axioms (and a confidentiality property) the consistency problem is decidable. It is even in PTIME [17] if we do not include the most general integrity axiom.

**Case studies** A prototype verification tool (SCARY) has been implemented (by G. Scerri), which is encouraging. For instance known and also new attacks have been found [31].

2.3 A summary of our research program

The preliminary results that are reported above encourage us to develop the approach further and experiment it at a larger scale. This includes the investigation of another logic that embeds indistinguishability properties (section 3), the design of modular proofs (section 4), the design of additional axioms, for instance axioms about new cryptographic primitives (section 5), experimentations on a larger scale (section 6) and considering side channels attacks (section ??).

3 Task 1: Indistinguishability properties

We have recently designed [8] another logic that embeds indistinguishability properties. This logic is simpler than the previous one, but it is not yet clear how easy (or difficult) it will be to automate.

We first need to study more examples, which will witness the relevance of the set of axioms that we have, or, to the contrary, require new axioms that we will have to design.

Then, the main task will be to design appropriate strategies and possibly to prove tractability results in the spirit of [17]. Depending on the outcome, we will (or not) implement another tool, based on this logic. If the logic turns out to be easily decidable, we need to carry experiments on practical case studies. If the logic turns out to be undecidable or too complex, we will focus more on the development of the SCARY tool.
4 Task 2: Proof (de)composition

Our method has some inherent composability properties: when a protocol $P_1$ is proved secure, adding more axioms will restrict the attacker’s capabilities, not increase them. Therefore, the protocol $P_1$ will remain secure.

If we add a new protocol $P_2$ that runs in parallel of the protocol $P_1$, we need however to revise the former security proof. $P_2$ introduces more traces, some of which may violate the security property. In general, proving $P_1$ and $P_2$ separately does not imply that, together, they are secure.

Still, our framework provides with a possible solution for composing the proofs. First observe that, if the axioms $A_1$ that are used in the security proof of $P_1$, are sound in the presence of a computational attacker that has an access to $P_2$ (as an oracle), then the security of $P_1$ holds for this stronger model of attacker. This implies in particular that, if in addition $P_2$ alone is secure, then $P_1$ and $P_2$ together are secure.

Now, proving that an axiom $A_1$ is sound with respect to an attacker that has access to $P_2$ as an oracle, could again benefit from our approach. We could see $A_1$ as a security property that has to be satisfied by $P_2$, this time considering a (simple) computational attacker.

In summary, we would have to prove that $\neg A_1 \cup A_2 \cup \theta_2$ is inconsistent for any constraint $\theta_2$ generated by a trace of $P_2$ and then the inconsistency of $\neg \phi \cup A_1 \cup A_2 \cup \theta_1$ for any constraint generated by a trace of $P_1$, where $\phi$ is the security property.

Though this looks easy in theory, there are some hidden difficulties. For instance, currently, the logical fragment in which the axioms are defined is different (and larger) than the fragment in which the security properties are defined. This feature is used in the design of efficient strategies. Also, once more, we need concrete examples and experiments to test the practical relevance of such an idea.

5 Task 3: Axiom design

Though a number of axioms are designed and proved to be sound in the computational model in [31], they only concern the classical properties of encryption. As for indistinguishability properties, very few axioms were considered in [8]. We need to design and prove axioms for other primitives, under various assumptions and in different logics.

One of the challenges, which would be interesting to investigate, is the case of exclusive-or. Indeed, [5, 32] (for instance) show that it is impossible
to design a computational soundness result for exclusive-or. This means that we cannot specify equationally (and independently of the security parameter) what an attacker can do. But what about what an attacker cannot do?

Another challenge concern the hash functions, for which similar impossibility results have been proved.

There are easier cases that should be investigated first, such as signatures, zero-knowledge proofs and possibly others.

6 Task 4: Automation and case studies

Although the consistency checks can often be completed in polynomial time, we need to complete such a check for every possible sequence of control states. In principle, we have to consider any interleaving of control states of the processes. Therefore, there are in principle exponentially many sequences of control states that have to be considered.

If we want to consider significant case studies, it is necessary to improve upon such a naive implementation. There are several directions that can be investigated. First, the deductions that are performed on some branch should be re-used (when relevant) in other branches. Sharing more deductions across branches may require us to modify the saturation strategy. In summary: we have to design an efficient saturation strategy for each trace, which also takes into account the possible further reuse of intermediate deductions. There is some hope here since, for instance, the same axioms and security property are used on each trace.

Another direction that could yield dramatic improvements are partial order reductions, such as in [28].

In the case of equivalence properties (see section 3), the problem is different, as all traces are folded in a single one [8]. However, the very folding process, as well as the proof strategies, deserves attention.

In the end, the goal is to find unexpected attacks on well-known protocols, which were proved using various tools, but with a weaker attacker model! We especially target widely deployed protocols such as Kerberos [29]. But we also expect to automatically analyse protocols that are currently beyond the scope of the exisitant automatic provers, because they use both complex primitives and equivalence properties.
7 Task 5: Beyond the computational attacker

In the section 4, we have seen a possible use of an attacker, who is stronger than the computational one (because it may have access to external oracles).

An advantage of our approach is that it works for any model of attacker. So far, we have only considered applications to the computational attacker, reducing the computational security to the inconsistency of a set of first-order formulas. Such an inconsistency result however implies something stronger: the security w.r.t. any model of attacker that satisfies the axioms. If we design the axioms in such a way that they are valid for other models of attackers, we will get security guarantees for such attackers.

As an example, we could consider an attacker who also observes execution time. Such an attacker will distinguish messages that require different computation times. We could design axioms that state the indistinguishability (w.r.t. computation time) of some basic operations and guard the other indistinguishability axioms by such indistinguishability computation times. In such a setting, an inconsistency proof would show the security in presence of an attacker who may observe the execution time. Conversely, a model would be a witness of an attack that exploits such observations.

Similar methods can be applied to the power consumption observation, for instance [27].

In this last task we will investigate security proofs for several such stronger attackers.

References


15