AI for securing communications

Laurent Vigneron
Université de Lorraine - LORIA, Nancy, France
laurent.vigneron@loria.fr

Abstract

Any organization, public or private, has to set up electronic services, for internal or external needs. These services have to guarantee a high level of security, either for protecting exchanged information, for authenticating the involved actors, or for guarantying the quality and correction of those services. Securing communications is in general provided by cryptographic protocols, but to prove that those protocols really achieve the required security properties is often very difficult. Some methods based on automated reasoning are defined to try to get those proofs, or to show some failures.

Keywords: cryptographic protocol, security property, automated reasoning.

Résumé

Les organisations, qu'elles soient publiques ou privées, mettent en place de plus en plus de services électroniques, que ce soit pour des besoins internes ou externes. Ces services doivent garantir un niveau de sécurité élevé, que ce soit pour protéger les informations échangées, authentifier les acteurs impliqués, ou garantir la qualité et la correction des services. Cette sécurisation des communications est en général assurée par des protocoles cryptographiques, mais démontrer que ces protocoles apportent une réelle sécurité est souvent difficile. Des méthodes de raisonnement automatique sont définies pour tenter d'apporter ces preuves, ou de montrer des failles.

AI for securing communications

Abstract

Any organization, public or private, has to set up electronic services, for internal or external needs. These services have to guarantee a high level of security, either for protecting exchanged information, for authenticating the involved actors, or for guarantying the quality and correction of those services. Securing communications is in general provided by cryptographic protocols, but to prove that those protocols really achieve the required security properties is often very difficult. Some methods based on automated reasoning are defined to try to get those proofs, or to show some failures.

Keywords: cryptographic protocol, security property, automated reasoning.

To communicate through computer networks is necessary for any company: between internal services, between the company and external entities, or customers, or prospects, or suppliers. Securing these communications is often necessary, but the main problem is to be sure that the means set up bring the required security level.

For example, consider the following communication between a company and a supplier:

Alice@mycompany → Bob@supplier: *"Please deliver 1t of Vanadium, payed via account 999@fidji."*

Bob@supplier → Alice@mycompany: "*Got the money! Delivery in 3 days.*"

For such a communication, the network used my have ears, so many security properties may be needed such as:

- Authentication: Bob has to be sure that this message really comes from Alice and MyCompany. Alice has to be sure that the answer comes from her supplier.
- Secrecy: Alice may not want that anybody (except Bob) knows anything about this order.
- Integrity: Information in the message has not to be modified (for example the quantity ordered, of the account number).
- Availability: Is Bob always available for answering? Alice has to know when to decide that the message never reached Bob, and then send it again, taking the risk to double the order.
- Non repudiation: Bob has to keep a proof of Alice's order, so that she could not deny having sent it.
- Anonymity: Alice may want that her company could not be identified in this transaction.

Security Protocols

Security properties are considered thanks to security protocols. They are kinds of small programs performing the exchange of some information between the involved actors, permitting to establish a secure communication between these actors. We use everyday many of such protocols: SSL/TLS for a secure navigation on the Web, Kerberos for securely accessing to some distant services, ssh for a distant connection on a server, AKA protocols for securing mobile communications, …

The design of a protocol has to take into account several constraints, such as the degree of opening of the network to be used (that is the capabilities of an intruder on this network). The messages composing a protocol may use cryptographic mechanisms (encryption algorithms using symmetric/public/private keys, hash functions), signatures or certificates. And they often require preliminary actions (distribution of keys, generation of certificates).

Example: NSPK Protocol

The Needham Schroeder Public Key (NSPK) protocol is a good example of authentication protocol. The following version has been designed so that the two actors (Alice and Bob) authenticate each other. Messages are encrypted with an asymmetric algorithm: a message is encrypted with the public key of the receiver; only he can decrypt it with his private key. And random numbers (called *nonces*) are used to distinguish several executions of the protocol.

Alice → Bob: $\{N_A,A\}PK_B$

Bob → Alice: $\{N_A,N_B\}PK_A$

Alice → Bob: $\{N_B\}PK_B$

The interpretation of this protocol is the following:

Alice sends to Bob a nonce (N_A) and her name (A) to say she want to communicate with him. The message is encrypted with the public key of Bob (PK_B). When Bob decrypts the message, he learns the two informations included. Then he answers by sending back the nonce of Alice (N_A), adding its own nonce (N_B), encrypted with the public key of Alice (PK_A). Alice knows that only Bob could compose this answer as he was the only one able to learn N_A. So in the last message she confirms to Bob the reception of N_B.

The mutual authentication between Alice and Bob seems to be guaranteed, however there is a failure: a dishonest agent Charlie can mislead Bob, pretending to be Alice; this is possible for Charlie by exploiting a communication with Alice. Here is the scenario of the attack:

Alice → Charlie: $\{N_A,A\}PK_C$ Charlie[Alice] → Bob: $\{N_A,A\}PK_B$

Bob → Alice: $\{N_A,N_B\}PK_A$

Alice → Charlie: $\{N_B\}PK_C$ Charlie[Alice] → Bob: $\{N_B\}PK_B$

In this attack, Charlie uses the information sent to him by Alice to contact Bob, pretending to be Alice (written Charlie[Alice]). As Bob answers to Alice, his message cannot be decrypted by Charlie and goes directly to Alice. Alice receives this message, decrypts it, and as it contains the nonce N_A that she has sent to Charlie, she has no way to guess that this message has not been composed by Charlie but by Bob. So by sending the answer, she learns the value of N_B to Charlie. Charlie then uses this information to make Bob believe that Alice is ready to discuss with him.

The problem in the NSPK protocol comes from the fact that, in the second message, there is no information about who composed it. A possible correction could be to replace the second message by:

Bob → Alice: $\{N_A,N_B,B\}PK_A$

This example shows that security protocols may look convincing, but their correctness needs to be formally proved.

Verification of Security Protocols

Defining methods for verifying security protocols is very difficult, and most of the time proving the correctness of a protocol is impossible. The analysis is often focused on the finding of attacks. But even in this case, the problem may be undecidable, for example if considering an unbounded number of combined sessions of the protocol and an unbounded number of messages that can be composed by an intruder (Durgin et al., 1999).

A consequence is that many verification methods have been designed for specific classes of protocols, or for restricted capabilities of the intruder, or for hypotheses limiting the number of sessions or the number of messages to be generated.

However, reasoning processes based on artificial intelligence may help defining efficient verification methods and even, in some rare cases, methods able to prove the correctness of a protocol.

For example, Chevalier & Vigneron (2004) have proved that combining the following four methods permits to get a complete process for a large class of protocols:

- to analyze precisely the knowledge evolution of each actor of the protocol,
- to analyze precisely the knowledge evolution of the intruder,
- to handle, thanks to constraints, the messages generated by the intruder for considering only useful ones,
- to handle how the intruder can exploit some actors to get additional useful knowledge.

We detail each of these methods in the next sections.

Analysis Of Actors' Knowledge Evolution

The analysis of a protocol requires a perfect understanding of the information used in each message, and in particular to check how an actor can compose each message he has to send. The following process recursively defines how an actor A can compose a message M at step i of the protocol from his current knowledge, depending on the form of the message (a concatenation of messages, an encryption, an application of a hash function, or a newly generated information).

compose(A,M,i) = t if M is already known as t by A at step i

compose(A,(M_1,M_2),i) = (*compose*(A,M_1,i),*compose*(A,M_2,i)) // concatenation

compose(A,{M}K,i) = {*compose*(A,M,i)}*compose*(A,K,i) // encryption

compose(A,h(M),i) = *compose*(A,h,i)(*compose*(A,M,i)) // hash function

compose(A,M,i) = random_number if M is a new information generated at step i

compose(A,M,i) = Fail else

To get a precise understanding of the knowledge evolution of each actor, we also have to analyze the received messages. The objective is to check in each message what information is already known by the actor (so he will verify the value of this information), and what information is learned (because new to the actor). The following recursive process tries to decompose a message M received by actor A at step i of the protocol, according to his current knowledge. It runs sometimes the *compose* process to check if a (part of a) message can be composed from the current knowledge.

expect(A,M,i) = *compose*(A,M,i)

expect(A,(M_1, M_2),i) = (*expect*(A,M_1,i),*expect*(A,M_2,i)) // concatenation

expect(A,{M}PK,i) = {*expect*(A,M,i)}inv(*compose*(A,SK,i)) // public encryption

expect(A,{M}SK,i) = {*expect*(A,M,i)}inv(*compose*(A,PK,i)) // private encryption

expect(A,{M}K,i) = {*expect*(A,M,i)}*compose*(A,K,i) // symmetric encryption

expect(A,M,i) = $x_{A,M,i}$ else // new knowledge

In this decomposition process, if a message is encrypted with a public key (PK), the actor needs to know the corresponding private key (SK) to decrypt it, and similarly for a message encrypted with a private key, the corresponding public key has to be known. Note that *inv* is a symbolic notation for the inverse of an asymmetric key, that is: inv(PK)=SK and inv(SK)=PK. In case of a symmetric encryption, as the same key is used to encrypt and to decrypt, it has to be known by the actor.

The result of the *expect* process is a skeleton of the received message, where unknown information is replaced by a new variable, for expressing that this new information is memorized.

Application to the NSPK protocol: considering the actor Alice, her initial knowledge is A, PK_A, SK_A, B, PK_B. And her participation to the protocol can be analyze by:

compose(Alice,{N_A,A}PK_B,1) = {n_1,A}PK_B // n_1 is a random number

expect(Alice,{N_A,N_B}PK_A,2) = {n_1,$x_{A,NB,2}$}inv(SK_A) // $x_{A,NB,2}$ stores the value of N_B

compose(Alice,{N_B}PK_B,3) = {$x_{A,NB,2}$}PK_B

And considering the actor Bob, his initial knowledge is B, PK_B, SK_B, A, PK_A. His participation to the protocol can be analyze by:

expect(Bob,{N_A,A}PK_B,1) = {$x_{B,NA,1}$,A}inv(SK_B) // $x_{B,NA,1}$ stores the value of N_A

compose(Bob,{N_A,N_B}PK_A,2) = {$x_{B,NA,1}$,n_2}PK_A // n_2 is a random number

expect(Bob,{N_B}PK_B,3) = {n_2}inv(SK_B)

This precise analysis of actors' knowledge in a protocol has several advantages: first, it permits to check that the protocol can be implemented (each actor is able to compose the messages he has to send); second, when composing a message, it permits to know where each used information comes from (initial knowledge or a previously received message); third, when receiving a message, we know exactly what information is already known and so can be checked, and what information is new and has to be stored. This is a very good guide for implementing a protocol.

Analysis Of Intruder's Knowledge Evolution

Verifying a protocol does not only mean to handle actors' knowledge, it also requires to determine what information can be learned by an intruder listening to the network. In the general case, an intruder car read all the messages posted over the network. So this is important to study each message of a protocol, to check if they can be decomposed by an intruder, i.e. an agent that does not participate to the communication. This analysis is done by the *expect* process described above, applied to a third agent Charlie, and based on his initial knowledge corresponding to all information that is public (like the names of the actors, their public keys).

Application to the NSPK protocol: considering the intruder Charlie (C), with initial knowledge A, B, C, PK_A, PK_B, PK_C, SK_C, the information he can learn by listing to a communication between Alice and Bob is:

expect(Charlie,{N_A,A}PK_B,1) = $x_{C,Msg1,1}$

expect(Charlie,{N_A,N_B}PK_A,2) = $x_{C,Msg2,2}$

expect(Charlie,{N_B}PK_B,3) = $x_{C,Msg3,3}$

This shows that an intruder cannot decompose any message of this protocol, because he does not know the private key of Alice or Bob. And he cannot compose himself any of those messages because they all contain at least one information that he does not know, the nonces. But knowing those messages may however be useful.

Handling Of Generated Messages

For studying the security of a protocol, we have to model the behavior of an intruder over the network. We have seen that, just by listening to the network, he can get some information. But an intruder is most of the time active over the network: he sends some messages, either in his own name, or impersonating other actors. The most complex phase is to model the construction of those messages, as with only a constant and a key, one can build infinitely many messages (example: C, {C}K, {{C}K}K, {{{C}K}K}K, …). For controlling this complexity, a lazy strategy can be used, as proposed by Basin (1999). It consists in building only messages that can be accepted by the actors of the protocol, i.e. that are well-formed. This well-formedness is given by the *expect* process, as for a given message, it describes the skeleton that can be checked by the receiver, and the parts that he will learn, so that can contain anything. The model of the intruder will therefor try to build messages with the right skeleton, according to the current knowledge of the intruder, and thanks to the *compose* and *expect* processes, it will study how each information is propagated in the next messages, with the objective that a communication will reach its end and generate an attack with respect to a security property.

Handling Of Puppet Actors

Analyzing a protocol means trying to find attacks in an official communication between honest actors. But listing to this official communication and trying to send alternative messages is not sufficient. Attacks often require to combine several sessions of a protocol: the official communication and additional ones in which the intruder may have a role. So, modeling the intruder behavior also consists in generating new protocol sessions, in which he will use either his real identity or impersonate some other actors. The objective of those additional sessions is to get a (part of a) message that will be useful for building a well-formed message to be used in the official communication. So a process has to be defined to run those alternative sessions of the protocol, with actors that the intruder will use as puppets.

The four processes described above represent important tasks for analyzing the correctness of a security protocol. The real complexity of the analysis is to combine them efficiently. This is the role of artificial intelligence: it will exploit the static analysis of the protocol (*compose* and *expect* processes) and the security properties to be checked, to build an engine modeling the capabilities of an intruder, that is building skeletons of messages that could be accepted by an official actor and then get the messages sent by this actor as responses for exploiting them in the next steps of the protocol. In the skeleton of messages, the parts that cannot be checked by the receiver are

constrained by the current knowledge of the intruder because later on, those parts will be propagated in further messages, and some of them may become checkable by an actor; the modeling engine will then have to verify that it can instantiate those lately checkable parts so that the message will be accepted by the receiver.
At the end of this process, if the modeling engine succeeds to build a complete communication falsifying a security property, this is an attack if the final set of constraints has a solution: all the skeletons of messages can be entirely filled thanks to the intruder's knowledge.

Verification engines, like the one described above, are more and more used by protocols' creators, because artificial intelligence permits to define efficient reasoning techniques, and therefor to analyze the protocols for finding attacks or trying to prove their correctness. The publication of verified protocols is essential for encouraging their implementation in a large number of applications, and brings much more confidence to the final users in numerical communications.
However, there are still some fields where users are not confident in numerical systems. An example is electronic voting.

A Special Case: Electronic Voting

Electronic voting via Internet is possible for simple votes, considering a very limited number of voters, thanks to many simple tools that do not consider seriously security problems. But if a secure system is required, finding a trusted tool may not be easy and organizing the vote may require technical skills. In addition, if we have to manage a very large number of voters, the right tool may be even more difficult to find. And note that for national elections, very few countries use electronic voting. This is partially due to a psychological barrier of the voters. But the main reason is that important elections need to guarantee the highest level of security, and this is very difficult to reach mainly because the threats are quite different from those of classical communications over a network. Here are some examples of specific threats:

- the vote server can be remotely attacked,
- the vote server can be physically attacked,
- the voting system may contain bugs or failures (accidental or intentional),
- the computer of a voter may be compromised.

Verifying a voting system consists in verifying the voting software itself, but also at runtime the handling of the vote server and of each voter's computer. Security properties to satisfy have to take all this into account:

- a vote has to be confidential: no one should know the vote of any voter; and a voter should not be able to prove the value of his vote;

- the voting process has to be honest and transparent: it has to be verifiable; each voter should have a proof that his vote is in the ballot box; all the ballots have to come from legitimate voters; the result of the election has to correspond to the ballots in the box;
- the voting system has to be available and accessible: any voter has to be able to vote, at any time during the voting period, and without requiring some technical skills.

Some secure voting systems exist: Civitas (Clarkson et al., 2008) Helios (Adida et al., 2009), Belenios (Cortier et al., 2019), … But they are not adapted for very large elections. However a system like Belenios is more and more used by associations and companies. Years passing, the confidence of voters into numerical systems will improve, the electronic voting systems will be more and more secure and efficient, so we can hope that national elections will become electronic in several years.

All this problematic of electronic voting is very well explained in Cortier & Gaudry (2022).

Conclusion

We have presented some of the main principles used for automatically analyzing security protocols. This is still a difficult task because of the complexity of the problem, and the variety of security properties to be satisfied according to the considered protocol. Reasoning techniques derived from artificial intelligence permit to define efficient processes, even if they often cannot obtain a complete certification of the protocols. They help to design more secure protocols and therefor to bring more confidence in numerical communications.

But this is a never ending research domain, as hundreds of security protocols are designed every year, covering an ever larger set of domains and security properties. And verifying the specification of a protocol is only a first step: the implementation of a specification should also be verified, as some technical choices may have been taken that could generate failures; there are many examples of such problems, like for TLS (Fiterau-Brostean et al., 2020).

Bibliography

- Adida, B., de Marneffe, O., Pereira, O. and Quisquater, J.-J. (2009). *Electing a University President Using Open-audit Voting: Analysis of Real-world Use of Helios.* In Proceedings of the conference on Electronic voting technology, Workshop on trustworthy elections, Montreal, Canada. https://www.heliosvoting.org/
- Basin, D. (1999). *Lazy Infinite-State Analysis of Security Protocols.* In Secure Networking – CQRE, Lecture Notes in Computer Science 1740, pages 30-42, Düsseldorf, Germany.

- Chevalier, Y. and Vigneron, L. (2004). *Strategy for Verifying Security Protocols with Unbounded Message Size.* Journal of Automated Software Engineering, 11(2): 141-166. Kluwer Academic Publishers.
- Clarkson, M. R., Chong, S. and Myers, A. C. (2008). *Civitas: Toward a Secure Voting System.* In Proceedings of IEEE Symposium on Security and Privacy, pages 354-368. https://www.cs.cornell.edu/projects/civitas/
- Cortier, V., Gaudry, P. and Glondu, S. (2019). *Belenios: A Simple Private and Verifiable Electronic Voting System.* In Foundations of Security, Protocols, and Equational Reasoning, pages 214-238. https://www.belenios.org
- Cortier, V. and Gaudry, P. (2022). *Le vote électronique – Les défis du secret et de la transparence.* Odile Jacob editor.
- Durgin, N., Lincoln, P., Mitchell, J. C. and Scedrov, A. (1999). *Undecidability of Bounded Security Protocols.* In FLOC's Workshop on Formal Methods and Security Protocols, Trento, Italy.
- Fiterau-Brostean, P., Jonsson, B., Merget, R., de Ruiter, J., Sagonas, K. and Somorovsky, J. (2020). *Analysis of DTLS Implementations Using Protocol State Fuzzing.* In: USENIX Security Symposium, pages 2523–2540, Boston, USA.