A Fast Method for Security Protocols Verification

Olga Siedlecka-Lamch[1](✉), Sabina Szymoniak[1], and Miroslaw Kurkowski[1,2]

[1] Institute of Computer and Information Sciences, Czestochowa University of Technology, Czestochowa, Poland
{olga.siedlecka,sabina.szymoniak}@icis.pcz.pl

[2] Institute of Computer Science, Cardinal St. Wyszyński University, Warsaw, Poland
m.kurkowski@uksw.edu.pl

Abstract. Internet communication is essential for everyone. Algorithms that decide about the correctness of this communication are protocols, and the central part of it that keeps all in safety are security protocols. Because every such program must be implemented and applied, errors are probable. That is why we need verification methods based on mathematical models, and we also need tools checking the new protocols, looking for undiscovered gaps. Existing verification tools and languages describing the protocols are not free of errors or imperfections. Sometimes they neglect some dependencies, and sometimes they are utterly redundant. We present in the article a formal model that we have recently developed. It describes the different behaviours and properties of security protocols. On the base of it, we implemented the tool that verifies many types of protocol, first of all, if they work and then if they meet the security requirements. At the end of the article, we provided a summary of our results with the results obtained from popular tool.

Keywords: Verification of security protocols · Security protocols properties · Model checking

1 Introduction

Internet communication affects all areas of our modern life. We use it for virtual conversation, business, shopping, and in our private life. In each of the situations mentioned above, we may wonder if our data is safe, how much potential intruders can learn. Does every Internet user realize what Internet communication is, how many operations hide under it?

The Internet was initially the domain of scientists subsidized by the army. From the very beginning, it was supposed to enable safe communication, even in

The project financed under the program of the Minister of Science and Higher Education under the name "Regional Initiative of Excellence" in 2019–2022 project number 020/RID/2018/19, the amount of financing 12,000,000 PLN.

the event of damage or attack of parts of the network. Many devices, algorithms and software decide about the operation of the modern network. Examples of these algorithms are communication protocols that deal with data exchange. Their most essential parts are security protocols. These are small algorithms consisting of several steps to guarantee correct verification of communicating pages, as well as security of transmitted data.

During these several decades of Internet existence, scientists invented and implemented many protocols, many of them turned out to be defective. Errors and gaps were detected, sometimes only after many years of use [18,21]. There was and still is a strong need to verify the mechanisms that determine the security of our data.

The verifications were initially carried out experimentally, but of course, there are no better methods than formal verification, allowing the analysis of a much broader spectrum of possible situations. There are several paths for verification. The first approach is deductive verification, the descriptions of which can be found in the works of Burrows, Abbadi and Needham [1,6]. Basin and Wollf used logic in their work [3]. Another approach was based on inductive methods [20,22]. Currently, most of the works on verification of security protocols focus on model checking [4,9,10,19].

For each of the developed methods, teams of researchers created dedicated tools, firstly to show their performance and effectiveness, and secondly to provide verification tools to the public. Here is a list of the most popular tools: Scyther [7], ProVerif [5], AVISPA [2], UPPAAL [8], VerICS [13], PathFinder [17].

The question arises whether the topic has not been exhausted with such a long list of methods and tools. We should emphasise that researchers still develop new protocols, adapt them, and existing tools are not free of defects [15]. We also care about the time of the verification, especially since most tools search the redundant space of generated actions and their combination, which not reflects reality.

The article presents our current approach to protocol verification. Our team has been working for years in this field, developed techniques related to automatic machine networks [16], verification using SAT tools [14], probabilistic verification [24], up to the current method. Our methods have also been used in practice to verify the MobInfoSec system, which is a distributed, modular, and configurable cryptographic access control system to sensitive information [23]. In the method described in the article, a tuple represents a step of the protocol. In every tuple, we distinguish elements related to the conditions that the user must meet to send a message, elements related to the message itself, and elements suggesting what knowledge the recipient has gained. This simple structure allows a quick analysis of whether the user can take the next step. If he did not get the right knowledge in previous tuples - he can not. In this way, the model is strictly limited and does not allow for the analysis of paths that do not occur in reality.

The article consists of five sections: introduction, description of an exemplary protocol, description of the formal method, experimental results and a summary.

2 Exemplary Security Protocol

In this part of the article, we show an example of the security protocol used in the MobInfoSec system, which enables encryption and sharing of confidential information to a group of connected users of mobile devices [12,23]. The system includes software and hardware. A team of researchers and programmers developed it for popular mobile devices available on the market. One of the most complex components that met the ORCON rules was a strong mutual authentication scheme between secret protection modules (SP). The system performs communication between multiple users. Each of them on its device has two modules: Module Secret Protection (SP) and Authentication Module (MU) (Fig. 1).

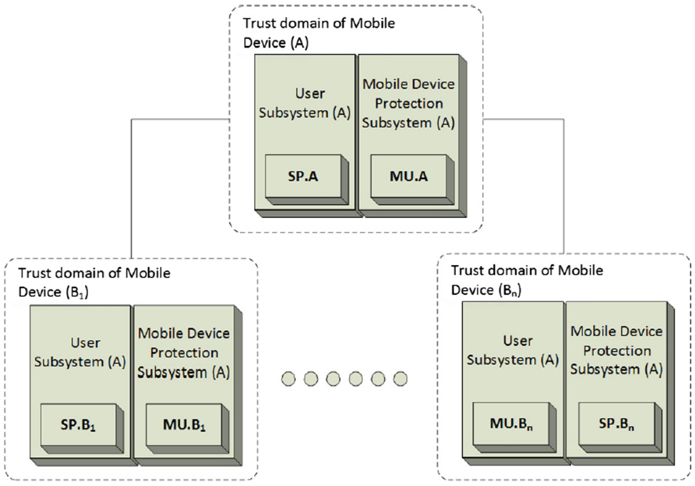
Trust domain of Mobile Device (A)
User Subsystem (A)
Mobile Device Protection Subsystem (A)
SP.A
MU.A
Trust domain of Mobile Device (B_1)
User Subsystem (A)
Mobile Device Protection Subsystem (A)
$SP.B_1$
$MU.B_1$
Trust domain of Mobile Device (B_n)
User Subsystem (A)
Mobile Device Protection Subsystem (A)
$MU.B_n$
$SP.B_n$

Fig. 1. Trust domains concept for different mobile devices

One of the communication points is taken as the initiator of the protocol (chairman - in the picture described by letter A) and should authenticate with other users (B_i); at the end, it establishes secure communication channels.

The entire communication protocol, which consists of five phases, is described in detail in the article [23], here we describe only the part responsible for authentication, which is a security protocol. For the description, we use the Common Language, which is the most popular form of protocol description appearing in the literature. We assume that before the security protocol, the keys of the $SP.A$ and $SP.B_i$ modules are activated, which is necessary to perform the cryptographic operations; there is also a request from $MU.A$ to $SP.A$ to generate a random number (nonce - a number used once).

α_1 $SP.A \rightarrow MU.A$: $\{N_{SP.A}, i(SP.A)\}$
α_2 $MU.A \rightarrow MU.B_i$: $\{N_{SP.A}, i(SP.A)\}$
α_3 $MU.B_i \rightarrow SP.B_i$: $\{N_{SP.A}, i(SP.A)\}$
α_4 $SP.B_i \rightarrow MU.B_i$: $\{\{N_{SP.B_i}, -k_{SP.B_i}, h(N_{SP.B_i}, N_{SP.A}, i(SP.A))\}_{-k_{SP.B_i}}\}_{+k_{SP.A}}$
α_5 $MU.B_i \rightarrow MU.A$: $\{\{N_{SP.B_i}, -k_{SP.B_i}, h(N_{SP.B_i}, N_{SP.A}, i(SP.A))\}_{-k_{SP.B_i}}\}_{+k_{SP.A}}$
α_6 $MU.A \rightarrow SP.A$: $\{\{N_{SP.B_i}, -k_{SP.B_i}, h(N_{SP.B_i}, N_{SP.A}, i(SP.A))\}_{-k_{SP.B_i}}\}_{+k_{SP.A}}$

where:

- $i(SP.A)$ - the identifier of the secret protection module of user A,
- N_X - random numbers - nonces generated by the module specified in subscript,
- $h(N_{SP.B_i}, N_{SP.A}, i(SP.A))$ - hash value calculated for a message $N_{SP.B_i}, N_{SP.A}, i(SP.A)$ using hash function h.
- $-k_SP.B_i$, $+k_{SP.A}$ - keys, public and private accordingly.

Let's follow the protocol step by step. The SP module belonging to the user A generates a random number $N_{SP.A}$ and, together with its identifier $i(SP.A)$, transmits it to the MU module (step α_1). Everything happens in one trusted domain. All this information is transferred to the user's B_i domain (step α_2), where it goes to the secret protection module from the authentication module (α_3). User B_i generates his nonce $N_{SP.B_i}$, adds previously obtained information and provides a hash function $h(N_{SP.B_i}, N_{SP.A}, i(SP.A))$, signs whole with his key $-kSP.B_i$ and encrypts with the user's A key $+k_{SP.A}$. The whole is sent in sequence from the SP to MU module in the user's B_i domain (α_4). Next to the user's A domain (α_5) and already internally from the MU module to the SP module. When the protocol is completed, the user's A module SP and each SP module of user B_i (for $i = 1, ..., n$) have confidential key materials $N_{SP.A}$ and $N_{SP.B_i}$ ($i = 1, ..., n$). On this basis, each party calculates the new symmetric key for independent, trusted channels.

3 Verification Method

In the paper [16], a mathematical model of a protocol's executions (correct runs) is transformed into a network of finite synchronous automata. Many protocol executions (the runs) were expressed as the computations in this network. The security problems were modelled as a problem of the reachability of desired (unsecured) states in the network. The computations in the network were subsequently encoded as propositional boolean formulas. SAT-solvers answer the question whether a valuation fulfilling the formula exists, and therefore whether an attack on the protocol exists. In the paper [16], several experimental results for untimed protocols were given. In some cases, they were better than those obtained during verification with the AVISPA Tool.

Now, we introduce a new approach. The double translation of the mathematical model of executions proposed in [16] brought along some redundancies, of which the following method is free. The obtained and described experimental results in the last section of the article are promising for the later development of this approach.

In the proposed method, we encode each step of a protocol execution in the form of a tuple (1) containing conditions for executing a given step and actions taking place during its execution. We can analyse many executions that are running in parallel. That is why we mark each step as α^i_j, where i is the number of the execution and j is the number of the protocol step. Each tuple takes the form:

$$\alpha^i_j = (Send, Rec, PreCond(S^i_j), S^i_j, PostKnow(S^i_j)), \tag{1}$$

where $Send$ is the sender, Rec is the receiver, S^i_j is the j-th step of the i-th protocol's execution, $PreCond(S^i_j)$ is the set of conditions that must be met for the step S^i_j to be executed and at the end $PostKnow(S^i_j)$ is the set of knowledge that is gained as a result of the step. As you can see, the order of the elements in the tuple clearly suggests the time sequence of individual actions.

We distinguish two types of $PreCond$, that must be held before a step:

1. G^X_U - represents the generation of new confidential information X (nonces, keys) by the user U (e.g. $G^{N_A}_A$ - the nonce N_A generated by the user A),
2. P^X_U - represents the requirement to have the given knowledge element X necessary for the user U to perform the given step U (e.g. $P^{N_B}_A$ - the user A must have the nonce N_B to perform the step).

In the $PostKnow$ set, you will find the knowledge gained as a result of performing the step, which will be marked as K^X_U - knowledge about the object X gained by the user U.

Example 1. Now we can represent steps of the security protocol part for MobInfoSec system in the form of tuples:

$$\begin{aligned} \alpha_1 = (&SP.A, MU.A, -sender\ \ and\ \ receiver \\ &\{P^{I_{SP.A}}_{SP.A}, G^{N_{SP.A}}_{SP.A}\}, -PreCond \\ &\{N_{SP.A}, i(SP.A)\}, -message \\ &\{K^{i(SP.A)}_{MU.A}, K^{N_{SP.A}}_{MU.A}\}) - PostKnow \end{aligned} \tag{2}$$

Since we can consider interleaving of different executions of the same protocol, we enter the designation $alpha^j_i$ where i is the step number, and j is the execution number. For clarity, let's also mark as $hash = h(N_{SP.B_i}, N_{SP.A}, i(SP.A))$ and as $message_{\alpha_4} = \{\{N_{SP.B_i}, -k_{SP.B_i}, h(N_{SP.B_i}, N_{SP.A}, i(SP.A))\}_{-k_{SP.B_i}}\}_{+k_{SP.A}}$. Thus, one arbitrary execution of the entire protocol looks as follows:

$$\begin{aligned}
\alpha_1^1 &= (SP.A, MU.A, && \{P_{SP.A}^{i(SP.A)}, G_{SP.A}^{N_{SP.A}}\},\\
& && \{N_{SP.A}, i(SP.A)\},\\
& && \{K_{MU.A}^{i(SP.A)}, K_{MU.A}^{N_{SP.A}}\})\\
\alpha_2^1 &= (MU.A, MU.B_i, && \{P_{MU.A}^{i(SP.A)}, P_{MU.A}^{N_{SP.A}}\},\\
& && \{N_{SP.A}, i(SP.A)\},\\
& && \{K_{MU.B_i}^{i(SP.A)}, K_{MU.B_i}^{N_{SP.A}}\})\\
\alpha_3^1 &= (MU.B_i, SP.B_i && \{P_{MU.B_i}^{i(SP.A)}, P_{MU.B_i}^{N_{SP.A}}\},\\
& && \{N_{SP.A}, i(SP.A)\},\\
& && \{K_{SP.B_i}^{i(SP.A)}, K_{SP.B_i}^{N_{SP.A}}\})\\
\alpha_4^1 &= (SP.B_i, MU.B_i && \{P_{SP.B_i}^{i(SP.A)}, P_{SP.B_i}^{N_{SP.A}}, P_{SP.B_i}^{+k_{SP.A}}, G_{SP.B_i}^{N_{SP.B_i}}, G_{SP.B_i}^{hash}\},\\
& && \{message_{\alpha_4}\},\\
& && \{K_{MU.B_i}^{message_{\alpha_4}}\})\\
\alpha_5^1 &= (MU.B_i, MU.A && \{P_{MU.B_i}^{message_{\alpha_4}}\},\\
& && \{message_{\alpha_4}\},\\
& && \{K_{MU.A}^{message_{\alpha_4}}\})\\
\alpha_5^1 &= (MU.A, SP.A && \{P_{MU.A}^{message_{\alpha_4}}\},\\
& && \{message_{\alpha_4}\},\\
& && \{K_{SP.A}^{N_{SP.B_i}}, K_{SP.A}^{hash}\})
\end{aligned} \qquad (3)$$

Let's explain the above formula on the example of the fourth tuple, which looks the most complex. The module $SP.Bi_i$ sends a message to the module $MU.B_i$. It needs the identifier $i(SP.A)$, the nonce $N_{SP.A}$, the key $+k_{SP.A}$, and must also generate the nonce $N_{SP.B}$, and the hash value. The module $SP.B_i$ sends the entire $message_{\alpha_4}$ to the module $MU.B_i$, but this second one can not do anything with message, $MU.B_i$ can only forward it to the user's A domain. The message reaches the right sender only in the sixth step, and then $SP.A$ can decipher the message and learn the nonce $N_{SP.B_i}$ and the hash value from it.

On this base, we can generate interlaces of many executions from many sessions, for example $\alpha_1^1, \alpha_1^2, \alpha_2^2, \alpha_2^1, ..., \alpha_5^1, \alpha_5^2$. How to check if the generated tuples are real and have the right to exist in actual protocol executions? We must define the so-called correct tuple. Let Π be the base space consisting of the users and their attributes (identifiers, nonces, cryptographic keys, etc.). We have to consider all the executions of the protocol in this space and all tuples for all executions. Under the set of all these tuples, we define a correct tuple represents the real executions of the protocol in the network.

Definition 1. *We call the sequence* $\mathfrak{s} = s_1, s_2, \ldots, s_p$ **a correct tuple** *iff the following conditions hold:*

1. *if* $s_i = S_j^k$ *for some* $j, k \leq p$ *then* $(j = 1 \vee \exists_{t<i}(s_t = S_{j-1}^k))$ *and* $PreCond(S_j^k) \subseteq \{s_1, \ldots, s_{i-1}\} \wedge PostKnow(S_j^k) \subseteq \{s_{i+1}, \ldots, s_p\}$,
2. *if* $s_i = G_U^X$, *then* $\forall_{t \neq i}(s_t \neq G_U^X)$,
3. *if* $s_i = P_U^X$, *then* $\exists_{t<i}(s_t = G_U^X \vee s_t = K_U^X)$.

The first point guarantees a proper dependence on the order of carrying out the individual steps of a given execution. Points second and third, ensure a dependence of the users' knowledge necessary to execute the individual steps. In particular, the second point guarantees that a given knowledge can be generated only once. The third point shows that a given user has some knowledge if he has generated it or obtained it as the result of one of the previous steps.

Thanks to the following theorem, we can reduce verification of considered security protocol for a given set of their executions to the analysis of the corresponding set of tuples that represents executions. Specifically, observe that there is an attack upon the protocol in the considered set of executions if there is an a correct tuple that represents attacking execution. An attacking execution contains an element in which the Intruder I learns secret information (for example $K_I^{N_{SP.A}}$), or can even finish the protocol by impersonating another user without being recognised. For the definition of the correctness of knowledge, see [14].

4 Experimental Results

The tool implemented by us (from now on referred to as an E-Ver from an efficient verifier) enabled modelling and generation of many security protocols, taking into account all the new assumptions described in the previous sections. Then, we compared the time results with the results of the known tool - ProVerif, with which we have not combined our method so far. We obtained all the results presented below on the same computer unit, equipped with an Intel Core i7 processor, 16 GB main memory and Ubuntu Linux operating system.

At the input, the tool accepts the protocol description defined in the ProToc language [11]. This description allows to include information about external and internal actions which are performed during protocol execution.

Table 1. Summary of the MobInfoSec security protocol executions

No.	Execution
1	$SP.A \rightarrow MU.A \rightarrow MU.B_1 \rightarrow SP.B_1$
2	$SP.A \rightarrow MU.A \rightarrow MU.B_2 \rightarrow SP.B_2$
3	$I \rightarrow MU.B_1 \rightarrow SP.B_1$
4	$I \rightarrow MU.B_2 \rightarrow SP.B_2$
5	$I_A \rightarrow MU.B_1 \rightarrow SP.B_1$
6	$I_A \rightarrow MU.B_2 \rightarrow SP.B_2$
7	$SP.A \rightarrow MU.A \rightarrow I$
8	$SP.A \rightarrow MU.A \rightarrow I_{B_1}$
9	$SP.A \rightarrow MU.A \rightarrow I_{B_2}$

Table 1 presents a summary of the MobInfoSec security protocol executions. We can find here all the communication possibilities between users A and B_i. In column 'Executions' is located information about the users according to the order they appear in the protocol. By A and B_i, we mark honest users. By I the Intruder is indicated. Designations I_A and I_{B_i} mean an Intruder impersonate honest users. For every execution in which Intruder take part, there are more possibilities for parameters (nonces, keys). The Intruder can use his own or other users parameters. we show below (example for one chairman and two other users).

Each of the nodes in our executions tree consists of five elements:

- execution's number,
- step's number,
- set of the needs (elements needed to execute a given step - $PreCond$),
- set of generated elements (elements generated in a given step - $PreCond$),
- set of knowledge (elements that increase the set of knowledge after a given step - $PostKnow$).

Elements which are located in mentioned sets are numbers referring to a cryptographic object used by users in protocol's executions. Thanks to such simplification, we can build a tree of nodes and verify a protocol faster than other tools. Also, our structure is smaller than in other tools.

Next, our tool tries to build a tree and find an attack. By following the methodology mentioned earlier, nodes that meet the imposed conditions are added to the tree. If the created path contains an attack, a tool returns information about finding an attack.

Let's show a summary of how our tool works for several popular protocols (Fig. 2.) like The Needham–Schroeder Symmetric Key Protocol ($NSSK$ [21] - It forms the basis for the Kerberos protocol), the Needham–Schroeder Public-Key Protocol corrected by Lowe ($NSPK_{LOWE}$ [21]). Wide-Mouth Frog protocol (WMF [6]) and Woo and Lam Pi protocol [25]. We examined three time parameters:

– node generation time (T_{NG}),
– time of finding attacks (T_{AF}),
– protocol checking time (T_{PC}).

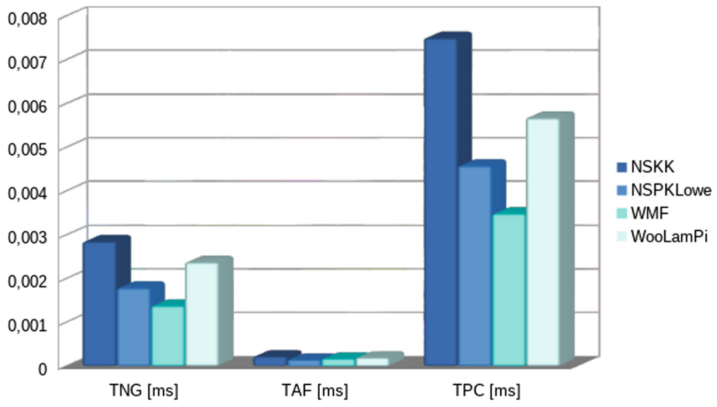

Fig. 2. Summary of protocols verification times in E-Ver in a graphical form

Please note that for all examined protocols an attack was found (normal attack or man-in-the-middle attack). For all protocols and all parameters obtained times were lower than 1 [ms] (Table 2).

Table 2. Summary of protocols verification times in E-Ver

Protocol	T_{NG} [ms]	T_{AF} [ms]	T_{PC} [ms]
NSPK	0.002853	0.000224	0.007503
NSPK$_{Lowe}$	0.001791	0.000145	0.004582
WMF	0.001382	0.000167	0.003488
WooLamPi	0.002378	0.000192	0.005677

Table 3. Comparison of E-Ver and ProVerif

Time [ms]	NSKK	Denning-Sacco
ProVerif	24	9
E-Ver	0.008659	0.005632

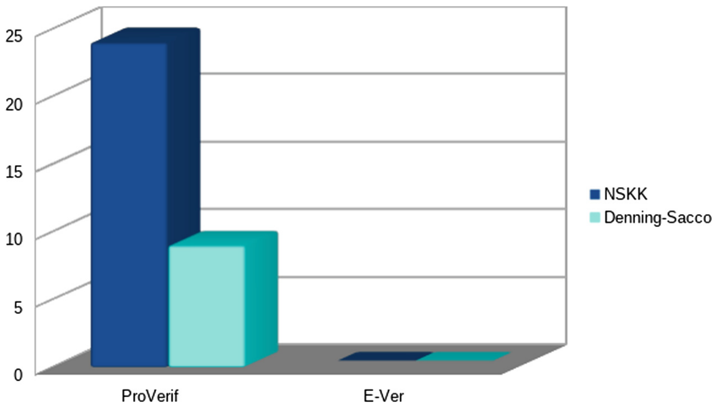

Fig. 3. Comparison of E-Ver and ProVerif in a graphical form

Next, we compared verification time with ProVerif (Table 3, Fig. 3). You can not specify the time of generating structures and the different stages for the tool ProVerif, so we limited the results to the total work time tool for the protocol.

5 Conclusion

The article presents the last work carried out by the authors in the field of verification of security protocols. Previous methods included many redundant operations and steps: building machines, testing with Sat-solver, and so on. The current formal model allows for a broad expression of the parameters of security protocols that are used nowadays. It allows to analyse the conditions needed to perform individual steps, takes into account the knowledge aspect, significantly limiting the searched space. The developed method allows to examine the interlaces of the same protocols for many sessions, to analyse the difficult-to-study Intruder model, which is Dolev-Yao.

The important thing is that for all of the protocols we examine, the time to detect attacks or search the whole tree in the absence of an attack is surprisingly small. Despite the size of the structure causing the exponential complexity of the algorithm, due to constraints imposed on attachable nodes (considering *PreCond* and *PostKnow*), the tree shrinks to real performances, which allows for quick analysis. Thanks to the structure built on constraints, we are also able to detect a situation in which the algorithm itself is incorrectly constructed, which is not always possible with the use of other tools.

References

1. Abadi, M., Blanchet, B., Fournet, C.: The applied pi calculus: mobile values, new names, and secure communication. J. ACM **65**(1), 1:1–1:41 (2018)
2. Armando, A., et al.: The AVISPA tool for the automated validation of internet security protocols and applications. In: Etessami, K., Rajamani, S.K. (eds.) CAV 2005. LNCS, vol. 3576, pp. 281–285. Springer, Heidelberg (2005). https://doi.org/10.1007/11513988_27
3. Basin, D., Clavel, M., Doser, J., Egea, M.: Automated analysis of security-design models. Inf. Softw. Technol. **51**(5), 815–831 (2009)
4. Basin, D., Cremers, C., Meadows, C.: Model checking security protocols. Handbook of Model Checking, pp. 727–762. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-10575-8_22
5. Blanchet, B.: Modeling and verifying security protocols with the applied pi calculus and ProVerif. Found. Trends Priv. Secur. **1**(1–2), 1–135 (2016)
6. Burrows, M., Abadi, M., Needham, R.: A logic of authentication. ACM Trans. Comput. Syst. **8**(1), 18–36 (1990)
7. Cremers, C., Mauw, S.: Operational Semantics and Verification of Security Protocols. Information Security and Cryptography. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-540-78636-8
8. David, A., Larsen, K.G., et al.: Uppaal SMC tutorial. Int. J. Softw. Tools Technol. Transfer (STTT) **17**(4), 397–415 (2015)
9. Dolev, D., Yao, A.: On the security of public key protocols. Technical report, Stanford, CA, USA (1981)
10. Gibson-Robinson, T., Kamil, A., Lowe, G.: Verifying layered security protocols. J. Comput. Secur. **23**(3), 259–307 (2015)
11. Grosser, A., Kurkowski, M., Piątkowski, J., Szymoniak, S.: ProToc—an universal language for security protocols specifications. In: Wiliński, A., El Fray, I., Pejaś, J. (eds.) Soft Computing in Computer and Information Science. AISC, vol. 342, pp. 237–248. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-15147-2_20
12. Hyla, T., Pejas, J., El Fray, I., Mackow, W., Chocianowicz, W., Szulga, M.: Sensitive information protection on mobile devices using general access structures. In: Proceedings of the Ninth International Conference on Systems, ICONS 2014, pp. 192–196. XPS (Xpert Publishing Services) (2014)
13. Kacprzak, M., et al.: Verics 2007 - a model checker for knowledge and real-time. Fundamenta Informaticae **85**(1–4), 313–328 (2008)
14. Kurkowski, M.: Formalne metody weryfikacji własności protokołów zabezpieczających w sieciach komputerowych. Informatyka - Akademicka Oficyna Wydawnicza EXIT, Akademicka Oficyna Wydawnicza Exit (2013)
15. Kurkowski, M., Kozakiewicz, A., Siedlecka-Lamch, O.: Some remarks on security protocols verification tools. In: Grzech, A., Świątek, J., Wilimowska, Z., Borzemski, L. (eds.) Information Systems Architecture and Technology: Proceedings of 37th International Conference on Information Systems Architecture and Technology – ISAT 2016 – Part II. AISC, vol. 522, pp. 65–75. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-46586-9_6
16. Kurkowski, M., Penczek, W.: Verifying security protocols modelled by networks of automata. Fundam. Inf. **79**(3–4), 453–471 (2007)
17. Kurkowski, M., Siedlecka-Lamch, O., Dudek, P.: Using backward induction techniques in (timed) security protocols verification. In: Saeed, K., Chaki, R., Cortesi, A., Wierzchoń, S. (eds.) CISIM 2013. LNCS, vol. 8104, pp. 265–276. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40925-7_25

18. Lowe, G.: An attack on the needham-schroeder public-key authentication protocol. Inf. Process. Lett. **56**(3), 131–133 (1995)
19. Lowe, G.: Breaking and fixing the needham-schroeder public-key protocol using FDR. In: Margaria, T., Steffen, B. (eds.) TACAS 1996. LNCS, vol. 1055, pp. 147–166. Springer, Heidelberg (1996). https://doi.org/10.1007/3-540-61042-1_43
20. Martina, J.E., Paulson, L.C.: Verifying multicast-based security protocols using the inductive method. Int. J. Inf. Secur. **14**(2), 187–204 (2015)
21. Needham, R.M., Schroeder, M.D.: Using encryption for authentication in large networks of computers. Commun. ACM **21**(12), 993–999 (1978)
22. Paulson, L.C.: Inductive analysis of the internet protocol TLS. ACM Trans. Inf. Syst. Secur. **2**(3), 332–351 (1999)
23. Siedlecka-Lamch, O., El Fray, I., Kurkowski, M., Pejaś, J.: Verification of mutual authentication protocol for MobInfoSec system. In: Saeed, K., Homenda, W. (eds.) CISIM 2015. LNCS, vol. 9339, pp. 461–474. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-24369-6_38
24. Siedlecka-Lamch, O., Kurkowski, M., Piatkowski, J.: Probabilistic model checking of security protocols without perfect cryptography assumption. In: Gaj, P., Kwiecień, A., Stera, P. (eds.) CN 2016. CCIS, vol. 608, pp. 107–117. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-39207-3_10
25. Woo, T., Lam, S.: A lesson on authentication protocol design. SIGOPS Oper. Syst. Rev. **28**(3), 24–37 (1994)