ISSN 2277-3061

Formal Methods in Information Security

AJAYI ADEBOWALE, NICULAE GOGA, OTUSILE OLUWABUKOLA, ABEL SAMUEL

Abstract

Formal methods use mathematical models for analysis and verification at any part of the program life-cycle. The use of formal methods is admitted, recommended, and sometimes prescribed in safety-and security-related standards dealing, e.g., with avionics, railways, nuclear energy, and secure information systems.

This paper describes the state of the art in the industrial use of formal methods ininformation security with a focus on verification of security protocols. Given the vast scope of available solutions, attention has been focused just on the most popular and most representative ones, without exhaustiveness claims.

We describe some of the highlights of our survey by presenting a series of industrial projects, and we draw some observations from these surveys and records of experience. Based on this, we discuss issues surrounding the industrial adoption of formal methods in security protocol engineering.

Keywords: Formal Methods; Information Security; protocol engineering*.*

Journal: INTERNATIONAL JOURNAL OF COMPUTERS & TECHNOLOGY

Vol 14 . No. 4

www.ijctonline.com , editorijctonline@gmail.com

Introduction

Despite the scores of formal methods proposed by various researchers, the methods are seldom used by software developers.The common perception among practitioners is that formal methods require too much effort (e.g., specification using temporal logic) and too much detail early in the development process, do not scale to industrial projects, are not cost-effective, and provide no clear benefits(James Kirby J. M., 1999).In an effort to investigate these perceptions this paper presents a review of experiences at applying formal methods to information security.

Formal methods in a broad sense are mathematically well-founded techniques designed to assist the development of complex computer-based systems; in principle, formal methods aim at building zero-defect systems, or at finding defects in existing systems, or at establishing that existing systems are zero-defect(Hubert Garavel, 2013).Formal methods can be seen as a scientist's reaction against empirical/organizational approaches, which sometimes focus more on the design process than on the product itself, and technical approaches that rely heavily on testing to detect (certain but not all) design and programming mistakes.

In many cases, computer automation delivers more flexible and reliable devices and infrastructures by enabling repetitive tasks previously done by humans, often in a sporadic manner, to be accomplished with precision and regularity. However, computer automation may also increase the risk of failures or malfunctioning, which may have severe consequences, especially for mission-critical and life-critical systems.

There are numerous examples of failures affecting computer-based systems. Regarding hardware-specific failures, one can mention the Pentium floating point division bug(1994) and the Cougar Point chipset flaw (2011), which costed Intel 475 million and one billion dollars, respectively. Regarding software-specific failures, the Therac 25 radiotherapy engine killed five persons in the 80s due to bad software design. Regarding large-scale infrastructures, the failure of the Denver airport automated baggage system (1994) delayed the airport's opening for 16 months with a cost overrun larger than 250 million dollars. This list is by no means complete, asevery week the Risks Digest forum reports new examples of risks to the public caused by computers and computer-based systems.

There are numerous reasons for failure of computer based systems: design errors, hardware faults, software bugs, security issues, performance issues among others. Formal methods have been hyped and touted as the best solution to each of these mentioned reasons for system failure. In an effort to ascertain the suitability of formal methods as a technique for ensuring proper functioning of computer based systems we review experience cases. With a focus on security issues we present a characterization and trend of development of formal method application to security protocol engineering.

LITERATURE REVIEW

2.1 Security issues

There are several possible definitions of security. According to (ISO (International Organization for Standardization). Information Technology – Z Formal Specification Notation – Syntax, Type System and Semantics, 2002), security is about "the protection of a system from malicious or accidental access, use, modification, destruction, or disclosure". More specific definitions exist; according to (ISO (International Organization for Standardization). Systems and Software Engineering – System Life Cycle Processes. International Standard 15288:2008, ISO/IEC, 2008), security concerns "all aspects related to defining, achieving, and maintaining confidentiality, integrity, availability, non-repudiation, accountability, authenticity, and reliability of a system". Other sources mention additional properties, such as anonymity, auditability, and privacy, for instance. Typical applications are access control systems, banking systems, smart cards, firewalls, operating systems, cryptographic protocols, voting machines, etc.

The contributions of formal methods to security are positively fruitful (for example, the historical survey of (Jeannette, 1998)). General-purpose formal techniques have been applied successfully to security issues; for instance, model checking tools enabled to find unknown attacks in security protocols, e.g., (Lowe and Gavin, 1995) for the Needham-Schroeder public-key authentication protocol (Roger Needham and Michael Schroeder, 1978)or (Guy and François , 2000) for the subscription and registration protocols of the Equicrypt conditional access and copyright protection system.

Additionally, dedicated formal methods have been developed to target security issues. Such methods include security-oriented formal notations, logics, and process calculi, as well as software tools for the automated analysis of secure protocols and systems; such tools take into account the particular characteristics of security problems to fight combinatorial explosion, for instance by joint use of theorem proving and model checking.

2.2 Protocol design and engineering

When designing a system, ensuring proper communications between the various parts of the system is an important issue. In many cases, it is feasible to address this issue in isolation from the rest of the system, by focusing on communications primarily and abstracting away all other aspects of the system. Using such an abstraction, the system is usually reduced to a set of agents interconnected with some communication network; these agents are running concurrently and using one or several protocolsto perform communication, synchronization, and/or co-operation towards common goals. In the particular case of cryptographic protocols, the system is abstracted to consider only certain aspects of communication related to information security, e.g., exchange of keys, transmission of encrypted data, etc.

ISSN 2277-3061

Protocol engineeringis the scientific methodology supporting protocol design. There has been a long-standing common history between formal methods and protocol design (Gregor von Bochmann, 2010). From the beginning, formal methods have been a core part of protocol engineering, and protocols have been a key application target for formal methods. This convergence of interests enabled major advances in theory and practice, a cross-fertilization that appears in several books, e.g. (Holzmann, 1992), (Sharp, 2008), in which protocol and formal aspects are intertwined.

Like any other communication protocol, a security protocol involves a set of actors, also called principals or agents, each one playing a protocol role and exchanging protocol messages with the other protocol actors. However, differently from normal communication protocols, security protocols are designed in order to reach their goals even in the presence of hostile actors who can eavesdrop and interfere with the communication of honest agents. For example, an attacker agent is normally assumed to be able to intercept and record protocol messages (passive attacker), and even alter, delete, insert, redirect, reorder, and reuse intercepted protocol messages, as well as freshly create and inject new messages (active attacker). The goals of the protocols are normally reached by using cryptography, which is why these protocols are also named cryptographic protocols.

The logic of a cryptographic protocol is often described abstractly and informally without getting into the details of cryptography. This informal description is also called the “Alice and Bob notation”, because protocol roles are usually identified by different uppercase letters and are associated to agent identities with evocative names in function of their roles, such as for example (A)lice for the first protocol participant, (B)ob for the second one, (E)ve for an eavesdropper and (M)allory for a malicious active attacker.

For example, the core part of the Needham & Schroeder (1978) public key mutual authentication protocol can be described in Alice and Bob notation as

1: A → B: {A,NA}KBpub

2: B → A: {NA,NB}KApub

3: A → B: {NB}KBpub

This protocol became notorious because of a flaw that was discovered by (Lowe, 1996)several years after its publication. The protocol aims at guaranteeing each participant about the identity of the other participant and at establishing a pair of shared secrets between them. A protocol description in Alice and Bob notation is a sequence of rules, each one describing a protocol message exchange in the form XY: M where X is the sender, Y is the intended recipient and M is the message. For example, the first rule specifies that agent Alice (A) sends to agent Bob (B) message {NA}KBpub. This writing stands for an encrypted pair where the components of the pair are Alice‟s identity A and a freshly generated nonce NA. The pair is encrypted with KBpub, that is Bob‟s public key. The (unreached) goal is that, after having completed the message exchange, Alice and Bob are certain about each other’s identity, and the nonces NA and NB have been shared between each other, but have been kept secret to anyone else.

Another semi-formal notation often used to represent the logic of security protocols is UML sequence diagrams. For example, the Needham & Schroeder (1978) protocol described abovecan be equivalently represented by the UML sequence diagram of Figure 1.

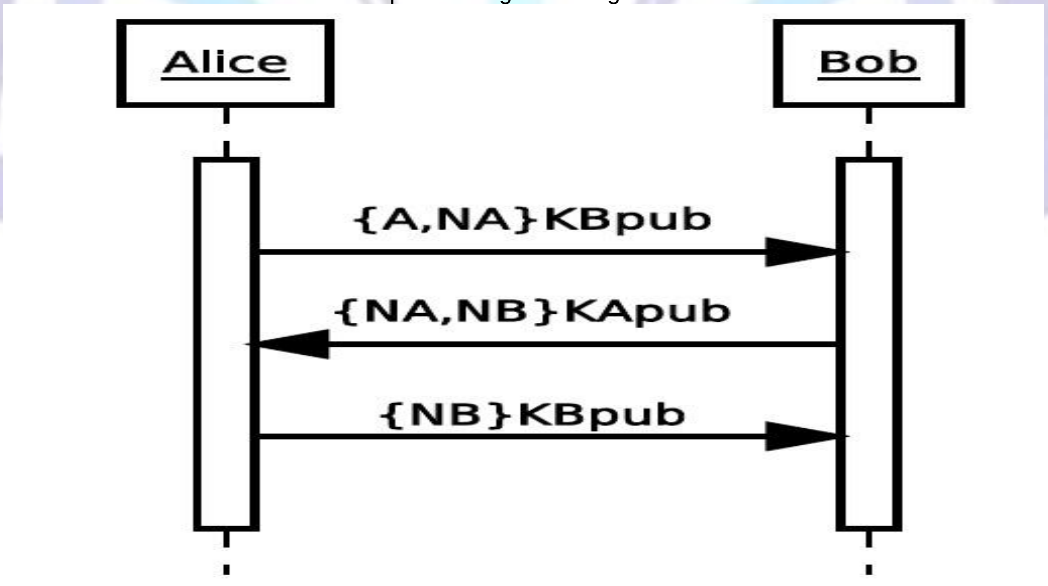

Figure 1 Needham-Schroeder public key protocol

Each actor has a vertical lifeline that extends from top to bottom, and exchanged messages are represented by arrows, whose caption is the content of the message. Both “Alice and Bob” notation and UML sequence diagrams are effective in representing a typical protocol scenario, while error handling or conditional execution are better represented by more formal and refined languages.

An attack on a security protocol is a protocol run scenario where the protocol does not reach the intended goal because of the hostile activity of an attacker. For example, if the protocol has been designed to ensure the secrecy of a secret datum D, an attack on the protocol is a protocol run in which an attacker gets the datum D (or partial knowledge on it). An attack is relevant only if it may occur with a non-negligible probability and using realistic resources. For example, the probability that an attacker who does not know a key guesses the value of the key is normally negligible because of the key length and because of its pseudo-random nature. The word “attack” often implicitly refers to run scenarios that may occur with non-negligible probability and using realistic resources.

Some attacks on security protocols exploit errors in protocol logic and are independent of details such as cryptographic algorithms or message encoding. These attacks can be described in terms of the abstract protocol descriptions shown above. For example, the attack discovered on the Needham-Schroeder protocol (Lowe, 1996) can be represented via the UML sequence diagram of Figure 2. The attack works out when a malicious agent Mallory (M) can convince Alice to exchange a nonce with him, so that he can use the messages from Alice to talk with Bob, while making Bob think that he is talking with Alice. Hence, Mallory is able to obtain Bob’s “secret” nonce that was intended for Alice.

Other attacks cannot be described abstractly in this way, because they depend on details or interactions of particular cryptosystems.

Attacks on security protocols can be categorized according to the protocol properties they break (e.g. secrecy, authentication). Moreover, in a more technical way they can be categorized by the categorization of the kinds of weaknesses or flaws they exploit (Gritzalis, S., Spinellis, D., & Sa, S., 1997)(Gritzalis, Spinellis & Sa, 1997 and Carlsen, 1994). According to the latter, the main classes can be described as follows.

Attacks based on Cryptographic Flaws exploit weaknesses in the cryptographic algorithms, by means of cryptanalysis, in order to break ideal cryptographic properties (e.g. determining the key used to encrypt a message from the analysis of some encrypted messages).

Attacks based on Cryptosystem-Related Flaws exploit protocol flaws that arise from the bad interaction between the protocol logic and specific properties of the chosen cryptographic algorithm. An example can be found in the Three-Pass protocol (Shamir, Rivest & Adleman, 1978) which requires the use of a commutative function (such as the XOR function) to perform encryption. Because of the reversible nature of the XOR function and of the way this function is used to encrypt the messages of the protocol, it is possible for an attacker to decrypt some encrypted data by XORing intercepted encrypted messages of the protocol session.

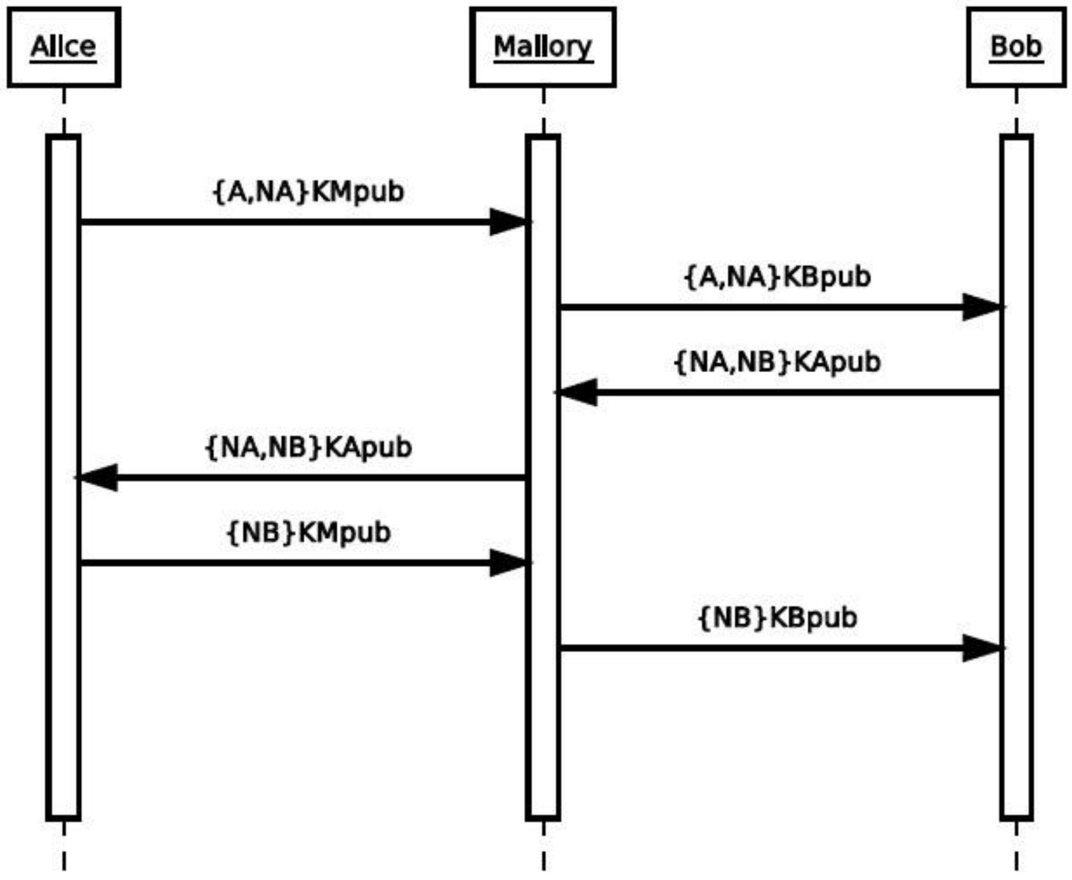

Figure 2: An attack on the Needham-Schroeder public key protocol

Another example can be found in some implementations of the widely deployed SSH protocol (Ylonen, 1996), where some bits of the encrypted plaintext can be recovered by an attacker, by injecting maliciously crafted packets when encryption is in CBC mode (Albrecht, Paterson & Watson, 2009). Attacks based on Elementary Flaws exploit protocol weaknesses that are considered elementary in the sense that the protocol completely disregards some kind of protection. For example, the attack described in Burrows, Abadi, & Needham (1990) on the CCITT X.509 protocol (Marschke, 1988) can be classified as an attack of this kind. Here, a message is encrypted and then a signature is applied to the encrypted part. Since the signature itself is not protected in any way, the attacker can simply replace it with the attacker‟s signature in order to appear as the originator of the message to the sender.

Attacks based on Freshness Flaws (or Replay Attacks) arise when one or more protocol participants cannot distinguish between a freshly generated message and an old one that is being reused. Thus, the attacker can simply replay an old message making the victim believe that it is a newly generated one. An example of a protocol containing this kind of flaw is the Needham & Schroeder (1978) secret key protocol, where the third message of the protocol can be replayed by an attacker, so that an old key (possibly compromised) is reused for the communication.

Guessing Attacks concern protocols that use messages containing predictable information encrypted with weak keys, either because poorly chosen (e.g. obtained by user passwords) or because generated by weak random number generators. These flaws can be exploited by analyzing old sessions of protocol runs in order to gain knowledge on the characteristics and on the internal status of pseudo-random generators. Another way to exploit these vulnerabilities is by performing dictionary-based attacks in order to guess keys, when the key space can be determined a priori (i.e. the most-likely combinations of bits forming the key can be determined). An example of a guessing attack is possible on the Needham & Schroeder (1978) secret key protocol, in addition to the reply attack explained above. The guessing attack can occur because keys are obtained from user-chosen passwords

Attacks based on Internal Action Flaws exploit protocol flaws that occur when an agent is unable to or misses to perform the appropriate actions required to securely process, receive, or send a message. Examples of such actions are: performing some required computation (such as a hash), performing checks aimed at verifying the integrity, freshness or authenticity of a received message, or providing required fields (such as a digest) inside a message. Often, these flaws are due to incomplete or wrong protocol specifications. An example of this flaw can be found in the Three-Pass protocol (Shamir, Rivest & Adleman, 1978) because, in the third step, the check that the received message must be encrypted is missing.

Oracle Attacks happen when an attacker can use protocol participants as “oracles”, in order to gain knowledge of some secret data, or in order to induce them to generate some data that the attacker could not generate on his own. Note that such data could be obtained by the attacker by interacting with one or more protocol sessions, even in parallel, while impersonating several agents. It is worth noting that these attacks may be possible because of the simultaneous presence of multiple protocols with unforeseen dependencies. Even if each protocol is safe, it may be that their combination is flawed because it enables some oracle attack. This may happen, for example, when the same keys are used by different protocols. For example, the Three-Pass protocol (Shamir, Rivest & Adleman, 1978 and Carlsen, 1994) is subject to this kind of attack in addition to the attack mentioned above.

Type Flaw Attacks happen when an attacker cheats a protocol agent by sending him a message of a type different than the expected one, and the receiver agent does not detect the type mismatch and uses message data in an unexpected way. An example of this attack is possible on the Otway-Rees protocol (Otway & Rees, 1987), where an attacker can reply parts of the first protocol message so that they are interpreted as a legitimate field of the last protocol message and in this way cheat Alice by inducing her to believe to have exchanged a secret key with Bob while in fact she has not. It is also possible to have attacks that exploit bugs in protocol implementations rather than bugs on protocols. For example, this occurs when an attacker exploits a divergence between the behavior of the implementation and the behavior prescribed by the protocol.

Attacks can furthermore be divided into two other broad categories: the attacks that can be performed by a passive attacker; and the attacks that require an active attacker. An attack performed by a passive attacker can be carried out without altering message exchange, simply by eavesdropping the exchanged messages. With this respect the Needham-Schroeder attack is an active attack, because the attacker must interact with the protocol agents by placing himself between the two. The most common forms of active attacks are the replay attack and the man in the middle (MITM) attack. In the former, the attacker replays an old message to an honest protocol actor that cannot distinguish between a freshly generated message and the old one. In the latter, the attacker intercepts all the messages exchanged by protocol actors (i.e. the attacker is in the middle of the communication), and relays intercepted messages and/or fabricates and injects new ones. In this way, actors are induced to believe that they are directly talking to each other, while, in the reality, the attacker is successfully impersonating the various protocol actors to the satisfaction of the other actors.

Security protocols normally reach their goals after a few message exchanges. However, despite this apparent simplicity, their logic flaws can be subtle and difficult to discover and to avoid during design. The difficulty comes not only from the bare use of cryptography, which is common to other cryptography applications. The extra complexity of security protocols comes from the unbounded number of different possible scenarios that arise from the uncontrolled behavior of dishonest agents, who are not constrained to behave according to the protocol rules, combined with the possibility to have concurrent protocol sessions with interleaved messages.

Without rigorous protocol models and automated tools for analyzing them it is difficult to consider all the possible attack scenarios of a given protocol. This has been demonstrated by the case of protocols that were designed by hand and discovered to be faulty only years after their publication, as it happened with the Needham-Schroeder authentication protocol. It is particularly significant that the attack on this protocol was discovered with the help of formal modeling and automated analysis (Lowe, 1996).

A rigorous, mathematically-based approach to modeling, analyzing and developing applications is called a formal method. The mathematical basis of formal methods opens the possibility to even prove facts about models. For example, a formal model of a security protocol and its environment can be mathematically proved to fulfill a given property, such as the inability of attackers to learn the protocol secret data. Of course, the results about protocol models are as significant as the model is. When a proof of correctness is obtained on a very abstract model, where many low-level details are abstracted away, attacks that rely on such low-level aspects cannot be excluded. On a more refined model, where such low-level details are taken into account, a proof of correctness can show those attacks are impossible. Unfortunately, as the complexity of models increases, a fully formal model and a proof of correctness become more difficult to achieve, and in any case a gap remains between formal models on one side and real protocols with real actors implemented in usual programming languages on the other side.

Formal methods can be supported by software tools. For example, tools can be available for searching possible attacks in protocol models or for building correctness proofs about models. When a formal method is supported by automatic tools, it is said to be an automated formal method.

Having automated formal methods is normally considered to be a must for formal method acceptability in production environments, because of the prohibitive level of expertise that is normally needed for using non-automated formal methods. One of the problems with automation is that as the complexity of models increases beyond a very abstract level, all the main questions about protocol correctness become undecidable, which prevents tools from always being able to come out with a proof of correctness automatically.

Traditionally, there have been two different approaches to rigorously model and analyze security protocols. On one hand are those models that use an algebraic view of cryptographic primitives, often referred to as “formal models”; on the other hand are those models that use a computational view of such primitives, generally referred to as “computational models”.

The algebraic view of cryptography is based on perfect encryption axioms: (1) The only way to decrypt encrypted data is to know the corresponding key; (2) Encrypted data do not reveal the key that was used to encrypt them; (3) There is sufficient redundancy in encrypted data, so that the decryption algorithm can detect whether a ciphertext was encrypted with the expected key; (4) There is no way to obtain original data from hashed data; (5) Different data are always hashed to different values; (6) Freshly generated data are always different from any existing data and not guessable (with non-negligible probability); (7) A private (resp. public) key does not reveal its public (resp. private) part.

Under these assumptions, cryptography can be modeled as an equational algebraic system, where terms represent data, constructors represent different encryption or data construction mechanisms, and destructors represent the corresponding decryption or data extraction mechanisms. For example, the constructor symEnc(M,k), the destructor symDec(X,k) and the equation symDec(symEnc(M,k),k) = M represent an algebraic system for symmetric encryption. As another example, the constructor H(M) represents hashing. The absence of a corresponding destructor represents non-invertibility of the cryptographic hash function.

Formal models allow simple and efficient reasoning about security protocol properties, because of their high level view on cryptography. However, there are two main drawbacks. One is that some cryptographic functions, mostly relying on bitwise operations, like for example XOR functions, are difficult to be represented in an equational algebraic model, or they could lead to undecidable models. The second one is that, because of the high abstraction level, there are several possible flaws that may be present in the protocol implementation, but that are not caught in the formal model. For example, this happens because the algorithms implementing a particular cryptographic function do not satisfy some of the ideal assumptions made: as an instance, any function generating n bit nonces cannot satisfy assumption (6) after 2n runs.

Computational models, in contrast, represent data as bit strings and use a probabilistic approach to allow some of the perfect encryption assumptions to be dropped. Assuming a bounded computational power (usually polynomial) for an attacker, the aim is to show that, under some constraints, the probability of an assumption being violated is negligible, that is, it is under an acceptable threshold. Technically, this is achieved by showing that an attacker cannot distinguish between encrypted data and true random data. Computational models can be used to deal with more cryptographic primitives, and to model more low level issues than formal models. However, they require more resources during protocol analysis, and usually the proofs are more difficult to be automated.

Historically, formal and computational models have evolved separately: the former focusing on the usage of basic cryptographic functions, and working at a higher level; the latter considering more implementation details and possible issues. Recent ongoing work starting from the seminal research of Abadi & Rogaway (2002), is trying to reconcile the two views and to relate them.

2.3 Recent Trends Of Development

When a new security protocol is being developed, design is the first phase where flaws can be introduced. Regardless of the severity of weaknesses that they can cause, these flaws are in general very critical, because they will apply to all deployed implementations. The use of formal modelling and formal verification is an effective way to mitigate this issue, because it improves the understanding of the protocol and guarantees its correctness, up to the detail level that is captured by the formal model. Although there is evidence that the use of formal methods improves quality of the outcome artefact, the cost-effectiveness of their adoption, especially when used only once in the design phase, is still uncertain (Woodcock, Larsen, Bicarregui & Fitzgerald, 2009). For this reason, best practices and design guidelines that protocol experts have defined after a careful analysis of the most common mistakes made in designing security protocols (e.g. Abadi & Needham, 1996 and Yafen, Wuu, & Ching-Wei, 2004) are another important resource for protocol designers.

Implementing a security protocol is another development phase where flaws can be introduced, due to divergences between the protocol specification and its implementing code. Such divergences are often caused by programming mistakes (e.g. OpenSSL Team, 2009) or by unfortunate interpretations of an ambiguous specification (e.g. Albrecht et al., 2009). Again, formally specifying the security protocol avoids interpretation errors, and in principle enables implementations to be (semi-automatically) derived, thus reducing the probability of introducing mistakes.

Other programming mistakes introduce so-called vulnerabilities. They are security problems that may affect any software that receives data from a public channel, not just security protocols. They derive from unhandled or unforeseen input data that may cause the protocol implementation to crash or to produce nasty effects on the host where the protocol implementation is running. Notable examples are stack overflows that can be exploited in order to run arbitrary code on the host where the software runs. Vulnerabilities do not directly imply a failure of reaching the protocol goals, but they can be exploited for violating other security policies or for compromising the whole host.

2.4 Automating Formal Protocol Analysis

When done by hand, formal analysis of security protocol models can be an error prone task, as often the security proofs require a large number of steps that are difficult to track manually.

The final goal of automating such techniques is to give formal evidence (i.e. a mathematical proof) of the fact that a protocol model satisfies or does not satisfy certain security properties under certain assumptions. The mathematics and technicalities of these techniques require specific expertise which may not be part of the background knowledge of security experts. Then, automation and user-friendliness of such techniques and tools are key issues in making them acceptable and useable in practice.

Research in this field has always paid highest attention to these issues. For example, as highlighted by Woodcock et al. (2009), automation in formal methods has recently improved so much that in 2006 the effort of automating the proof of correctness for the Mondex project, a smartcard-based electronic cash system, was just 10% of the effort that would have been required in 1999. In fact, in 1999 the system was proven correct by hand; in 2006 eight independent research groups were able to get automatic proofs for the same system.

2.4.1. Logics of Beliefs (BAN Logic)

Logics of beliefs (e.g. Burrows, Abadi, & Needham, 1990) are very high-level models for reasoning on security protocols where perfect cryptography is assumed, and only some logical properties are tracked. Technically, they are modal logic systems designed to represent the beliefs of protocol actors during protocol executions. For example, the formula “P believes X” means that actor P believes that predicate X is true, in the sense that P has enough elements to rightly conclude that X is true. Thus, P may behave like X was true. Another formula is “fresh(X)”, which means X is fresh data, that is it has not been previously used in the protocol (not even in previous sessions). If X is a newly created nonce, fresh(X) is assumed to hold just after its creation and the actor who just created the nonce believes it fresh (P believes fresh(X)). When a protocol session executes and messages are exchanged, the set of beliefs and other predicates for each actor updates accordingly. This is described in belief logics by a set of inference rules that can be used to deduce legitimate beliefs. Security properties can then be expressed by some predicates on actors' beliefs that must hold at the end of a protocol session.

More specifically, a protocol is described by using the Alice and Bob notation, and assumptions about initial beliefs of each actor are stated. For example, before protocol execution, Alice may already trust a particular server, or be known to be the legitimate owner of a particular key pair. Then, protocol execution is analyzed, so that beliefs of actors are updated after every message is exchanged. For example, after $A \rightarrow B$: M has been executed, “A believes A said M” is true, where “A said M” means that A has sent a message including M. The BAN logic inference rules also allow new facts to be inferred from those that are already known. At the end of the protocol, actors' beliefs are compared with the expected ones: if they do not match, then the desired property is not satisfied.

BAN logic can be very useful in protocol comparisons. Indeed, it is possible to compare the initial assumptions required by different protocols that achieve the same security goals. Moreover, it may be possible to find redundancies in the protocol, for example when an actor is induced by a received message to believe a fact that the actor already knows.

A BAN logic model can be verified by means of theorem proving. The theorem proving approach consists of building a mathematical proof in a formal deduction system in order to show that a desired security property holds in the given model. Since manually building formal proofs is a difficult and error-prone task, automation can be particularly effective. Some proof assistants, called theorem provers, can automatically solve simple lemmas, and they also interactively help the user in searching for the complete proof. Some of them, called automatic theorem provers, can even find complete proofs automatically

2.4.2 Dolev-Yao Models

The Dolev & Yao (1983) model of security protocols is currently widely used in many research works (e.g. Hui & Lowe, 2001, Abadi & Gordon, 1998, Bengtson, Bhargavan, Fournet, Gordon, & Maffeis, 2008, Fábrega, Herzog, & Guttman, 1999, Durgin, Lincoln, & Mitchell, 2004, Chen, Su, Liu, Xiao, 2010) and implemented in many protocol verification tools (e.g. Durante, Sisto, & Valenzano, 2003, Viganò, 2006, Mödersheim & Viganò, 2009, Blanchet, 2001). Since its introduction, it has gained popularity because it is a simple, high level modeling framework, yet effective in modeling common protocol features and reasoning about many common security properties.

Like logics of beliefs this way of modeling protocols stems from an abstract algebraic view of cryptography, but it is more accurate because it tracks exchanged messages rather than beliefs. In the Dolev-Yao model, a protocol is described by a discrete state-transition system made of communicating parallel sub-systems representing protocol actors. The transition system of each actor describes the actor behavior in terms of sent and received messages. For honest agents this is the behavior prescribed by the protocol, while for attacker agents it can be any behavior.

Many network models with apparently different features can be considered, but in general they can be shown to be equivalent, because one system can be encoded, or simulated, by the others. For example, one possible network model proposed by Schneider (1996) considers the network and the attacker as separate sub-systems, where the attacker is a protocol participant that has special actions, such as forging and eavesdropping messages. Other models (Ryan & Schneider, 2000) instead consider the intruder as the medium itself. Finally, association models (Voydock & Kent, 1983) group all securely connected actors into one “super-actor”, leading back to the basic idea of just having “super-actors” communicating only through an untrusted network. Usually, one chooses the network model that will make proving a particular security property as easy as possible.

2.4.3. Representing Dolev-Yao Models as Process Calculi

Among the different formalisms that have been developed to specify security protocols at the Dolev-Yao level, process calculi are among the most used ones, hence they are taken as an example here. Specifically, a variant of the applied pi calculus, which is one of the most user-friendly process calculi, is shown as an example.

Process calculi are algebraic systems designed to represent communicating processes and their exchanged data, with a well-defined formal semantics. Because of their abstraction level, they are a good tool for formally describing communication protocols in the Dolev-Yao style.

For example, Spi calculus is a process calculus for security protocols (Abadi & Gordon, 1998) that has been designed as a security-aware extension of the pi-calculus (Milner, 1999). It is close to a programming language, in that it enables explicit representation of input/output operations as well as checks on received messages. This is amenable to the derivation of implementations, since operations in the specification can be mapped into corresponding operations in the implementation.

The applied pi calculus (Abadi & Fournet, 2001) is similar to the Spi calculus but it is based on an equational system where constructors, destructors and equations can themselves be specified, making the language easily extensible. The slightly extended and machine readable version of the applied pi calculus as it is accepted by a popular verification tool (Blanchet, 2009) is presented as an example below.

The applied pi calculus syntax is composed of terms, formally defined in Table 1, and processes, formally defined in Table 2. A term can be an atomic name (e.g. “a“), a variable (e.g. “x”) that can be bound to any other term once, or a (constructor) function application f(M1, M2, …, Mn). For each constructor, corresponding destructor functions and equations can be defined separately, so as to specify an equational system that models the properties of cryptographic operations. For convenience, a syntactic sugar is introduced for tuples of terms, denoted by (M1, M2, …, Mn), which can be encoded within the standard syntax by corresponding constructor and destructor functions.

Table 1

M, N ::=	**Terms**
a, b, c	name
x, y, z	variable
f(M1, M2, …, Mn)	function application

Table 2

P, Q ::=	Processes
0	null process
P | Q	parallel composition
!P	replication
new a; P	restriction
if M = N then P else Q	conditional
let x = g(M1, M2, …, Mn) in P else Q	destructor application
in(c,x); P	message input
out(c,M); P	message output
event f(M1, M2, …, Mn); P	auxiliary authenticity event

2.4.4. Automated Verification of Dolev-Yao Models by Theorem Proving

Security protocols expressed as Dolev-Yao models can be verified by theorem proving. Essentially, the protocol specification is translated into a logic system where facts such as "message M has been transmitted" or "the attacker may know message M" are represented. Therefore, proving that a security property holds amounts to proving that a particular assertion (for example, "the attacker may know secret S") cannot be derived in the formal system. However, due to the undecidability of security properties such as secrecy and authentication in Dolev-Yao models (Comon & Shmatikov, 2002), it is not possible to develop an automatic procedure that in finite time and space always decides correctly whether or not a security property holds for any given protocol model.

Three main possible ways to cope with this undecidability problem have been explored. One is restricting the model (e.g. by bounding the number of parallel sessions), so as to make it decidable. This approach, which has been extensively used for state exploration techniques, has the drawback of reducing the generality of the results. A second possibility is to make the verification process interactive, i.e. not fully automatic. Here the obvious drawback is that, although libraries of reusable theorems can reduce the degree of interactivity, user expertise in using a theorem prover is needed. Moreover, there is no guarantee of eventually getting a result. A third possibility, that suffers from the same latter limitation, is using semi-decision procedures, which are automatic but may not terminate, or may terminate without giving a result, or may give an uncertain result. This approach has been adopted, for example, by Song, Berezin, & Perrig (2001) and Blanchet (2001). In particular, the ProVerif tool (Blanchet, 2001, Blanchet, 2009) is also publicly available and is one of the most used tools in the class of automatic theorem provers for Dolev-Yao models. It is based on a Prolog engine and accepts protocol descriptions expressed either directly by means of Prolog rules that are added to the ProVerif formal system, or in the more user-friendly applied pi calculus described in Table 1 and Table 2, which is automatically translated into Prolog rules. ProVerif does not terminate in some cases. When it terminates, different outcomes are possible. ProVerif may come up with a proof of correctness, in which case the result is not affected by uncertainty, i.e. the protocol model has been proved correct under the assumptions made. ProVerif may sometimes terminate without being able to prove anything. In this case, however, it is possible that ProVerif indicates potential attacks on the protocol. This ability is particularly useful to understand why the protocol is (possibly) flawed and is generally not provided by other theorem provers. Note however that the attacks produced by ProVerif may be false positives, because of some approximations it uses. That said, on most protocols the tool terminates giving useful results, which makes it one of the most used automated tools now available for verifying security protocols.

3.0 CONCLUSION

This paper has investigated the much touted perceptions about the adoption of formal methods. Using information security as the application domain with a focus on security protocol engineering, we have presented a characterization and trend of development of formal methods application to security protocol engineering.

This paper has introduced the main problems that have to be addressed for engineering security protocols and how formal methods and related automated tools can now help protocol designers and implementers to improve quality. The paper has stressed the progress that has been made in this field in the last years. While in the past, formal security proofs required so high levels of expertise that only few researchers and field experts could develop them, the availability of fully automated tools has now enabled a wider number of protocol developers to get advantage of formal methods for security protocols.

Two indicators of field maturity show that there is still some research work to be done. The first indicator is the presence of standard frameworks or methodologies. As a matter of facts, no commonly agreed standard exists in protocol specification or verification. Very few de-facto standards for generic security protocols exist (for example: the ISO/IEC CD 29128 (2011))and investigating the extent of their adoption is not a trivial issue.

The second indicator is the usage of formal methods in the release of a security protocol standard. As the tool support is often not completely automatic and user-friendly, and it requires some expertise and implies a steep learning curve, formal methods are usually deemed too expensive for the average security protocol standard. The result is that often standards are released without the support of formal methods, and the latter are applied after the standard is published (see the AVISPA project, Viganò, 2006), making it hard to fix issues highlighted by their application. It is believable that as soon as more automatic and user-friendly tools will emerge, and the presence of networks of computers will become even more pervasive and dependable, formal methods will be adopted, in order to fulfill the call for some form of certification of correctness.

REFERENCES

1. Abadi, M., & Needham, R. (1996). Prudent engineering practice for cryptographic protocols. IEEE Transactions on Software Engineering, 22 , 122-136.
2. Abadi, M., & Gordon, A. D. (1998). A Calculus for Cryptographic Protocols: The Spi Calculus. Research Report 149.
3. Abadi, M., & Fournet, C. (2001). Mobile values, new names, and secure communication. In Symposium on Principles of Programming Languages (pp. 104-115).
4. Abadi, M., & Rogaway, P. (2002). Reconciling two views of cryptography (The computational soundness of formal encryption). Journal of Cryptology, 15 (2), 103-127.
5. Albrecht M.R., Watson G.J. & Paterson K.G. (2009). Plaintext Recovery Attacks Against SSH. In IEEE Symposium on Security and Privacy (pp. 16-26).
6. Alfredo P., Davide P. and Riccardo S.(2011)Automated Formal Methods for Security Protocol EngineeringDip. di Automatica e Informatica, Politecnico di Torino, Italy
7. Bengtson, J., Bhargavan, K., Fournet, C., Gordon, A. D., & Maffeis, S. (2008). Refinement Types for Secure Implementations. In IEEE Computer Security Foundations Symposium (pp. 17-32).
8. Blanchet, B. (2001). An Efficient Cryptographic Protocol Verifier Based on Prolog Rules. In IEEE Computer Security Foundations Workshop (pp. 82-96).
9. Blanchet, B. (2009). Automatic Verification of Correspondences for Security Protocols. Journal of Computer Security, 17 (4), 363-434.
10. Burrows, M., Abadi, M., & Needham, R. (1990). A Logic of Authentication. ACM Transactions on Computer Systems, 8 (1), 18-36.
11. Carlsen, U. (1994). Cryptographic protocol flaws: know your enemy. In IEEE Computer Security Foundations Workshop (pp. 192-200).
12. Chen, Q., Su, K., Liu, C., Xiao, Y., (2010). Automatic Verification of Web Service Protocols for Epistemic Specifications under Dolev-Yao Model. In International Conference on Service Sciences (pp. 49-54). Dolev, D., & Yao, A. C.-C. (1983). On the security of public key protocols. IEEE Transactions on Information Theory, 29 (2), 198-207.
13. Comon, H., & Shmatikov, V. (2002). Is it Possible to Decide whether a Cryptographic Protocol is Secure or Not? Journal of Telecommunications and Information Technology, 4/2002, 5-15.
14. Durante, L., Sisto, R., & Valenzano, A. (2003). Automatic testing equivalence verification of spi calculus specifications. ACM Transactions on Software Engineering and Methodology, 12 (2), 222-284.
15. Durgin, N. A., Lincoln, P., & Mitchell, J. C. (2004). Multiset rewriting and the complexity of bounded security protocols. Journal of Computer Security, 12 (2), 247-311.
16. Fábrega, F. J. T., Herzog, J. C., & Guttman, J. D. (1999). Strand Spaces: Proving Security Protocols Correct. Journal of Computer Security, 7 (2/3), 191-230.
17. Hubert Garavel, D. S. (2013). *Formal Methods for Safe and Secure Computers Systems.* 24, rue Lamartine: Altros.
18. Hui, M. L., & Lowe, G. (2001). Fault-preserving simplifying transformations for security protocols. Journal of Computer Security, 9 (1/2), 3-46.
19. James Kirby, J. M. (1999). Applying Formal Methods to an Information Security Device: An Experience Report. HASE '99.
20. Milner, R. (1999). Communicating and Mobile Systems: the Pi-Calculus. Cambridge University Press.
21. Mödersheim, S., & Viganò, L., (2009). The Open-Source Fixed-Point Model Checker for Symbolic Analysis of Security Protocols. In Foundations of Security Analysis and Design (pp. 166-194).
22. Song, D. X., Berezin, S., & Perrig, A. (2001). Athena: A Novel Approach to Efficient Automatic Security Protocol Analysis. Journal of Computer Security, 9 (1/2), 47-74.
23. Schneider, S. (1996). Security properties and CSP. In IEEE Symposium on Security and Privacy (pp. 174-187).
24. Viganò, L. (2006). Automated Security Protocol Analysis with the AVISPA Tool. Electronic Notes on Theoretical Computer Science, 155, 61-86.
25. Voydock, V. L., & Kent, S. T. (1983). Security mechanisms in high-level network protocols. ACM Computing Surveys, 15 (2), 135-171.
26. **http://en.wikipedia.org/wiki/Denver_International_Airport#Automated_baggage_system retrieved 25th of October 2014**

27. Cougar Point chipset flaw**http://en.wikipedia.org/wiki/Sandy_Bridge#Cougar_Point_chipset_flaw retrieved 25th of October 2014**
28. Therac-25. Available at**http://en.wikipedia.org/wiki/Therac-25 retrieved 25th of October 25, 2014**
29. Pentium FDIV bug. Available at **http://en.wikipedia.org/wiki/Pentium_FDIV_bug retrieved 25th of October 25, 2014**
30. List of software bugs. Available at **http://en.wikipedia.org/wiki/List_of_software_bugs retrieved 25th of October 25, 2014**
31. **The Risks Digest (2014) Available at http://catless.ncl.ac.uk/risks**
32. **Safety Critical List (2014). Available athttp://www.cs.york.ac.uk/hise/sc list arc.php**