Probably Secure Channels in SPS language

Omar Saad Almousa
Amman Arab University
omaralmousa@aau.edu.jo

Abstract

Security Protocol Specification language (SPS) is a formal Alice-and-Bob style language for security protocols. SPS is translated not only to formal models, but also to implementations. The formal definition of SPS is made in the style of Dolev-Yao intruder, and it supports several types of channels including secure and insecure. In this paper, we introduce an extension to the SPS that generalizes these two channel types into *probabilistic* channels, i.e., the secrecy of such channels is a probability value ranging from $[0-1]$. We define the semantics for the extended SPS by a translation to probabilistic automata. We also give some further possible directions for this extension.

Keywords—Formal verification, Protocol security, Dolev-Yao intruder, Secure channels, Probabilistic automata.

I. Introduction

Security Protocol Specification language (SPS hereafter) [1] extends AnB [2] and specifies security protocols in Alice-and-Bob notation (aka protocol narrations). SPS is designed for modelling eID authentication protocols for the FutureID project that aims at building comprehensive, flexible, privacy-aware and ubiquitously usable identity management infrastructure for Europe [1]. Protocol specification in SPS is mainly used for formal verification and implementation generation. Targeted formal verification is done by connecting the specification to back-end model checking tools like OFMC [3], SATMC [4], CL-Atse[5] via a translation to AVISPA IF language [6] and to static analysis tools such as Proverif [7] via a translation to applied-π [8]. To that end, we define SPS semantics by a translation to an intermediate stage that we call *operational strands* extended from [9]. The formal semantics of operational strands is defined by a translation to an infinite state transition system [10]. Our formal model is based on the well-known Dolev-Yao intruder model [12]. Dolev-Yao model is widely used in formal protocol verification tools such as the aforementioned. In Dolev-Yao model the intruder controls the whole network. It can be seen as he is staying in the center of a star network with all protocol participants around [13]. Protocol participants exchange message only through the intruder that he can forward, intercept, change and obviously read (learn). Our approach of modelling *probably* secure channels can be seen as altering the placement of the intruder from the center of a network to any peripheral site that is connected, however, to all other participants, i.e., all participants are pairwise connected including the intruder. Figure 1 sketches the two views of the intruder.

We extend SPS language with *probabilistic* secure channels. This extension generalizes the existing secure channels, and models scenarios where channels are not totally secure, i.e., there is an opportunity to break their security with a certain probability. Our contribution is to enable the modelling and later the verification of the whole spectrum of channels from totally secure to completely insecure ones. We also model the probability of the intruder towards messages whether he forwards or blocks them. Finally, we extended SPS with the notion of lossy messages. The rest of this paper is organized as follows: in Section II we give a minimized version of SPS syntax for the purposes of this paper. We also give the semantics of SPS in a glance in Section III, and discuss attack states in Section IV. Finally we conclude in Section V.

[1] http://www.futureid.eu/

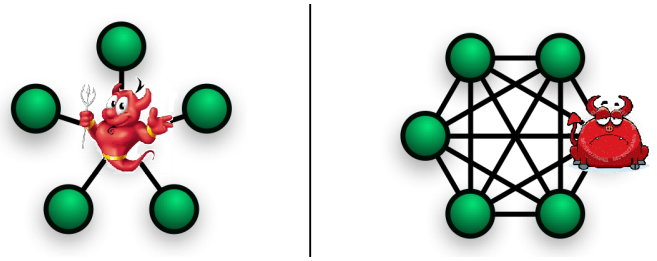

Fig. 1. Dolev-Yao Intruder: (left) Classical, (right) Weakened

II. SPS Syntax

SPS syntax inherits its simplicity and intuitiveness from Alice-and-Bob notation. In this paper, we give a minimal representation of SPS without excluding any of the required aspects for the purpose of this article. We also exclude the fine details of the SPS semantics. The full details of SPS can be found in [14]. We focus on the contribution of this paper i.e., introducing probabilistic channels in secrecy, intruder blocking and "lossiness". For protocol participants A and B, integers $l, m, n > 0$, operator $\mathsf{op} \in \{>, <, \leq, \geq\}$, probability values $p, q, r, v \in [0, 1]$, message msg and list of message msgs; the minimized SPS syntax is:

$$
\begin{array}{lcl}
\mathrm{SPS} & ::= & know_1; \ldots know_l; \\
 & & act_1; \ldots; act_m; \\
 & & goal_1; \ldots; goal_n \\
act_i & ::= & A \overset{(p,q,r)}{\rightarrow} B : \ \mathrm{msg} \\
know_j & ::= & A : \mathrm{msgs} \\
goal_k & ::= & \mathsf{secret}(A, B, \mathrm{msg}) \ \mathsf{op} \ v \\
 & & \mathsf{auth}(A, B, \mathrm{msg}) \ \mathsf{op} \ v
\end{array}
$$

A. Specification Sections

As seen in the syntax, an SPS protocol specification consists of three sections, namely knowledge, action, and goals.

Knowledge In this section we specify the *initial knowledge* of each participant to enable him to execute the protocol. The initial knowledge is a list of messages that the participant uses in composing sent messages or decomposing received ones. For example, $A : A, B, g, X$ says that the initial knowledge of participant A is his identity A, the identity of another participant B,and two values g and X.

Actions In this section we specify the actions that agents perform in a protocol, e.g., the messages that the protocol participants exchange or fresh data they create. Messages are exchanged in a token-passing mode, i.e. the receiver of the last message is the sender of the next one. The action $A \xrightarrow{(p,q)} B : \mathsf{exp}(g, X)$ says that the participant A sends the message $\mathsf{exp}(g, X)$ to the participant B via the annotated channel $\xrightarrow{(p,q)}$, where p is the probability that the message is delivered securely (from agent *A* to agent *B* without being intercepted nor blocked by intruder i), and q is the probability of the intruder forwarding the message or blocking it. Setting q is in a sense a weakening of the intruder, but we may want to analyse scenarios in which we know more about the intruder behavior.

Goals In this section we specify what are the goals a protocol is meant to achieve. In SPS we support two types of built in goals, namely secrecy (secret) and authentication (auth) goals. In general, the semantics of SPS supports the formalization of goals in the geometric fragment of [15] as shown in a previous result [16]. In this paper, we extend these goals by attaching a relational expression to enable the verification of a goal with a certain given probability. For instance, the goal $\mathsf{secret}(A, B, msg) > 0.8$ is achieved if there is no attack trace for which its probability is ≥ 0.2. We are considering attack traces since the semantics of SPS (as we show shortly) is defined by a translation to an infinite-state transition system. Reachable states represent what agents can perform. Goals are defined as events and accordingly *attack* states are defined as predicates over states; to enable verification tools to check the reachability of attack states. Further explanation is given later in this document.

III. SPS Semantics

The core of SPS semantics is the message model that we shortly present since this is not the scope of this article. For that reason, we exclude some details about message derivation and checking; the shortened presentation will be sufficient for the purpose of this paper, and the interested reader may refer to [10].

A. Message Model

We define SPS message model as a tuple: $(\Sigma, V, \approx)$ consisting of:

- Σ: a countable set of identifiers consisting of (a) constants denoted by Σ_0 representing the protocol constants, (b) constructors denoted by Σ_c such as scrypt and sign and (c) destructors denoted by Σ_d such as dscryptand open. We refer to (b) and (c) as public operators (Σ_p) in the sense that all participants can use them (including the intruder who we call i), so $\Sigma_p = \Sigma_c \cup \Sigma_d$. Table I lists the public operators for our minimized presentation.
- V: a countable set of protocol variables. We distinguish a special subset of V, namely V_A to refer to the agents names (roles). We use the term participant for the agent who is participating in a protocol. An example of a non-participant agent is a certificate authority that participants use its certificates without being participating in the protocol.
  A term t over Σ and a set of variables A denoted by $t \in \mathcal{T}_\Sigma(A)$ is a variable $t \in A$ or has the form $f(t_1, t_2, \ldots, t_n)$ where t_i are terms, $f \in \Sigma$ and of arity n. A *ground term* is a term t without variables denoted by $t \in \mathcal{T}_\Sigma$, i.e., $A = \emptyset$.
- $\approx$: a congruence relation over terms $\mathcal{T}_\Sigma$. For instance we have $\mathsf{dscrypt}(k, \mathsf{scrypt}(k, m)) \approx m$ to reflect the symmetric decryption of an encrypted term(message).

TABLE I. SPS Public Operators Σ_p

Description	Constructor(Σ_c)	Destructor(Σ_d)
Asymmetric encryption	crypt$(\cdot,\cdot)$	dcrypt$(\cdot,\cdot)$
Symmetric encryption	scrypt$(\cdot,\cdot)$	dscrypt$(\cdot,\cdot)$
Signature	sign$(\cdot,\cdot)$	open$(\cdot)$
Cryptographic hash	hash$(\cdot)$	
Keyed hash	mac$(\cdot,\cdot)$	

Moreover, we define a relation $\vdash$ between a knowledge M and a term t, denoted by $M \vdash t$ intuitively means that t is derivable from M. $\vdash$ is defined below as the least set closed under the following rules:

$$\frac{}{M \vdash m}\ (Axiom)\quad m \in M$$

$$\frac{M \vdash t}{M \vdash s}\ (Equivalence)\quad s \approx t$$

$$\frac{M \vdash t_1\ \ldots\ M \vdash t_n}{M \vdash f(t_1, \ldots, t_n)}\ (Composition)\quad f \in \Sigma_p$$

One can read $\vdash$ as inference rule in the form of if-then statements. For example, (Composition) means for any function $f \in \Sigma_p$, if $M \vdash t_1\ \ldots\ t_n$ then $M \vdash f(t_1\ \ldots\ M \vdash t_n)$. We use the operators specified in Table I to compose messages.

B. Strands

To define SPS semantics, we translate it into an infinite-state transition system through an intermediate translation to *operational* strands. The objective of translation to *operational* strands is to provide an intermediate stage from which further translation can be made to different input languages for back-end tools such as AVISPA IF [6] for AVIPSA tool chain (OFMC, SATMC,CL-Atse), applied-π for ProVerif [7] and

Horn clauses for SPASS [17]. It is also suitable for code generation (our SPS compiler translates to Java and JavaScript). This enhances the principle of specify-once-verify-many. See Figure 2. Operational strands definition is based on strands introduced in [9]. An operational strand is a sequence of actions that an agent performs during a protocol execution. It shows how an agent executes the protocol including composing and decomposing of exchanged messages. Strands define the behavior of participants of a protocol by means of an infinite-state transition system defined by an initial state and a transition relation on states. Reachable states represent what the system can do.

C. Translating to Strands

We define a translation function TR that translates from APS specification into a set of strands, i.e., a strand per participant agent. For a specification $Spec$ in which the first participant in the actions section (the initiator) is I, the set of strands is defined as:

$$\mathsf{TR}(Spec) = \{\mathsf{tr}(A, I, Spec) | A \ is \ a \ praticipant \ in \ Spec\}$$

Note that in the definition of $\mathsf{TR}(\cdot)$ we use the function tr that produces a strand for each participant. $\mathsf{tr}(A, I, Spec)$ translates the specification $Spec$ into the strand of the participant A. More precisely $\mathsf{tr}(A, B, Spec) \rightarrow Strand$ takes two agents, A represents to whom the translation is made, so we run tr for all agents participating in a protocol. The second agent B specifies which agent is *active*. If we consider protocols as token passing from an agent to another, *active* agent is the one holding that token. For that we used the initiator I in the initial call of tr. We define tr in Table II in which we use the bullet $\bullet$ to represent an argument that does not matter in tr case. Next, we explain the cases of tr referring to Table II:

1) The first case of tr presents knowledge filtering per participant, so we start the participant's strand with his initial knowledge.
2) This case translates for A who is active and sending msg to B.
3) This case translates for A when B is active and sending msg to A.
4) In this case, we translate for A when B is active and sending msg to any other agent C.
5) In this case, tr translates secrecy goal for A regardless who is active.
6) In this case, we translate authentication goal for A requesting authentication of B on msg regardless who is active.
7) In this case, we translate authentication goal for B, when A is requesting authentication of B on msg regardless who is active.
8) This is a generic case in which we try to generate a strand of any participant ($\bullet$). B is active, and C sends a message msg to any participant. Generally speaking, if $B \neq C$ then this causes an error; since the participant is taking the action (C) is not the active participant (B). Which contradicts the natural flow of messages that the sender of a message is the sender of the next one. This means that the protocol is not following a token passing form which leads to an error.
9) The last case represents the end of specification, i.e., nothing more to translate.

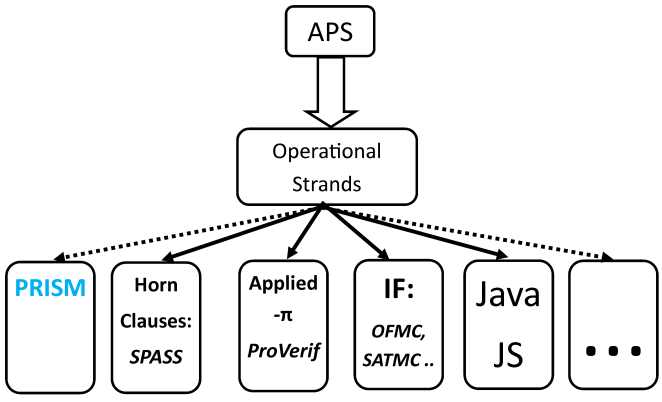

Fig. 2. SPS, Operational strands and other languages

D. Strands as Probabilistic Automata

We define the semantics of strands with presence of probabilities on channels as a probabilistic automata as in [18] based on [19] in which initial state is represented by the set of operational strands for all participants (*instantiated* with concrete values). A state in our transition relation is a tuple of three sets, namely a set of strands S, a set of messages representing the intruder knowledge M, and a set of events E. Events can be seen as regular messages but sent on a special channels. We denote the state by: $(S; M; E)$. Note that we either have strands as a multi-set or we force the use of a fresh session identifier for each strand. We define the initial state of our automata as follows:

Initial state As stated earlier the initial state is given by the set all participants' strands after instantiating variables with concrete fresh values for the required number of sessions. This instantiation includes the possible substitution of agents that are not honest by specification by the intruder or concrete *honest* agents. We distinguish two types of agents, honest agents that can not be instantiated with the intruder and *roles* (variable agents V_A) that can be instantiated with any concrete agent including the intruder.

Probabilistic transition relation We define a probabilistic transition relation over states on basis of the probability values given on channels (i.e., *p,q* above where p is the probability that an exchanged message is delivered securely and q is the probability of the intruder forwards it) as follows (recall that we translate to a probabilistic distributions over states, and we use $\mathcal{P}$ to represent one of them) :

a) Non-probabilistic case: This case represents the classical Dolev-Yao model in which the channel is completely insecure and the intruder is in full control of the network. Then it is all upon his decision to forward a message, change it or even block it. This is covered by these transition rules:

TABLE II. tr FUNCTION

(1)	$\mathsf{tr}(A, B, (K : \text{Rest}))$ where M is the initial of A in the knowledge section K	=	$M : \mathsf{tr}(A, B, \text{Rest})$
(2)	$\mathsf{tr}(A, A, (A \overset{(p,q)}{\rightarrow} B : \text{msg}).\text{Rest})$	=	$\mathsf{snd}(A, B, (p, q), \text{msg}).\mathsf{tr}(A, B, \text{Rest})$
(3)	$\mathsf{tr}(A, B, (B \overset{(p,q)}{\rightarrow} A : \text{msg}).\text{Rest})$	=	$\mathsf{rcv}(A, B, (p, q), \text{msg}).\mathsf{tr}(A, A, \text{Rest})$
(4)	$\mathsf{tr}(A, B, (B \overset{(p,q)}{\rightarrow} C : \text{msg}).\text{Rest})$	=	$\mathsf{tr}(A, C, \text{Rest})$
(5)	$\mathsf{tr}(A, \bullet, (\mathsf{secret}(A, B, \text{msg})\mathsf{comp}).\text{Rest})$	=	$\mathsf{secret}(A, B, \text{msg})\mathsf{comp}.\mathsf{tr}(A, \bullet, \text{Rest})$
(6)	$\mathsf{tr}(A, \bullet, (\mathsf{auth}(A, B, \text{msg})\mathsf{comp}).\text{Rest})$	=	$\mathsf{EndAuth}(A, B, \text{msg})\mathsf{comp}.\mathsf{tr}(A, \bullet, \text{Rest})$
(7)	$\mathsf{tr}(B, \bullet, (\mathsf{auth}(A, B, \text{msg})\mathsf{comp}).\text{Rest}$	=	$\mathsf{beginAuth}(A, B, \text{msg})\mathsf{comp}.\mathsf{tr}(B, \bullet, \text{Rest})$
(8)	$\mathsf{tr}(\bullet, B, (C \overset{(p,q)}{\rightarrow} \bullet).\text{Rest})$	=	error
(9)	$\mathsf{tr}(A, B, \epsilon)$	=	ϵ

$(\{\mathsf{snd}(A, B, (p, q), \text{msg}).rest\} \cup S; M; E) \Rightarrow \mathcal{P}$
and $\mathcal{P}$ is the following probability distribution over states
$\mathcal{P}((\{rest\} \cup S; M \cup \{\text{msg}\}; E)) = 1$

$(\{\mathsf{rcv}(A, B, x)\}.rest \cup S; M; E) \Rightarrow \mathcal{P}'$
$\mathcal{P}'((\{\sigma(rest)\} \cup S; M; E))= 1$
where $\sigma = [x \mapsto \text{msg}]$

b) Channel secrecy: The probability of channels secrecy is given, but not the probability of intruder decision (to block or forward the sent message). The transition rules for this case are:

$(\{\mathsf{snd}(A, B, p, \text{msg}).rest\} \cup S; M; E) \Rightarrow \mathcal{P}$
and $\mathcal{P}$ is the following probability distribution over states

$\mathcal{P}((\{rest\} \cup S; M; E \cup \{\mathsf{snd}(A, B, p, \text{msg})\})) = p$
$\mathcal{P}((\{rest\} \cup S; M \cup \{\text{msg}\}; E)) = 1 - p$

$(\{\mathsf{rcv}(A, B, x)\}.rest \cup S; M; E \cup \{\mathsf{snd}(A, B, p, \text{msg})) \Rightarrow \mathcal{P}'$
$\mathcal{P}'((\{\sigma(rest)\} \cup S; M; E)) = 1$
where $\sigma = [x \mapsto \text{msg}]$

c) Intruder decision: In this case we add the probability of the intruder decision of forwarding a message or blockin it. This introduces a strong control over the intruder, since by specifying q we give the probability of the intruder decision (p remains as the probability of the channel secrecy). This is the least realistic case. However, it models the situation where we have some prior knowledge about the intruder's behaviour. The transition rules for this ase are:

$(\{\mathsf{snd}(A, B, (p, q), \text{msg}).rest\} \cup S; M; E) \Rightarrow \mathcal{P}$
where $\mathcal{P}$ is the following probability distribution over states
$\mathcal{P}((\{rest\} \cup S; M; E)) = p$
$\mathcal{P}((\{rest\} \cup S; M \cup \{\text{msg}\}; E)) = (1 - p) * q$
$\mathcal{P}((\{rest\} \cup S \setminus \{\mathsf{rcv}(B, A, \text{msg})\}; M; E)) = (1 - p) * (1 - q)$

$(\{\mathsf{rcv}(A, B, x)\}.rest \cup S; M; E) \Rightarrow \mathcal{P}'$
$\mathcal{P}'((\{\sigma(rest)\} \cup S; M; E)) = 1$
where $\sigma = [x \mapsto \text{msg}]$

d) Lossy Channels: In here we model lossy channels, a message sent over a lossy channel with a certain probability may be lost and not received by neither the intended recipent nor the intruder. One can regard lossy channels as noisy channels with certain probability of message delivery failure say *r*. The rules for this case are:

$(\{\mathsf{snd}(A, B, (p, q, r), m).rest\} \cup S; M; E) \Rightarrow \mathcal{P}$
$\mathcal{P}((\{rest\} \cup S; M; E)) = r$
$\mathcal{P}((\{rest\} \cup S; M; E \cup \{\mathsf{snd}(A, B, (-), m)\})) = (1 - r) * p$
$\mathcal{P}((\{rest\} \cup S; M \cup \{m\}; E)) = (1 - r) * (1 - p)$
$\mathcal{P}(otherwise) = 0$
$(\{\mathsf{rcv}(A, B, \bullet, x)\}.rest \cup S; M; E \cup \{\mathsf{snd}(B, A, \bullet, m)\}) \Rightarrow \mathcal{P}'$
$\mathcal{P}'((\{\sigma(rest)\} \cup S; M; E)) = 1$
where $\sigma = [x \mapsto \text{msg}]$
$\mathcal{P}'(otherwise) = 0$

Global case We generalize all previous cases in one global case depicted in Figure 3.

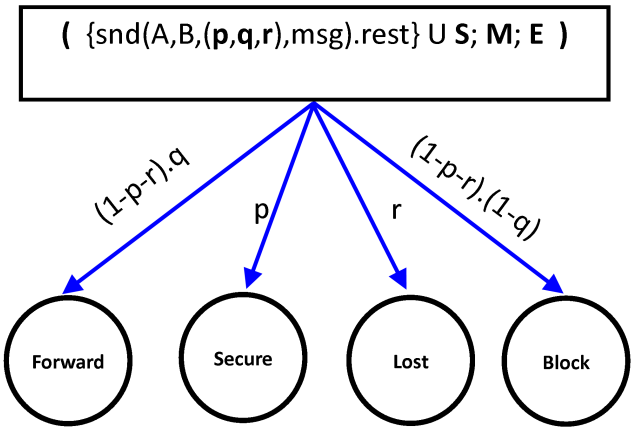

Fig. 3. *p-q-r* Channel Behavior

IV. ATTACK STATES

Attack states are defined according to protocol specification goals. More specifically they are given as predicates over states. As our goals are probability measured, we define the trace probability of any state as the minimum probability in that trace. For a state s, the trace probability of s is denoted by $\mathcal{P}^{trace}_{(s)}$. Then for any reachable state, if it is an attack. Now, we define an attack state to be a secrecy

attack state or an authentication attack state (or of course both). A state $s = (S; M; E)$ is a secrecy attack state iff $(\mathsf{secret}(A, B, \mathrm{msg})\mathsf{comp}) \in E \wedge m \in M \wedge A \neq \mathsf{i} \wedge B \neq \mathsf{i}$ and $(\mathcal{P}_{(s)}^{trace}\ \neg\mathsf{comp})$. The negation of comp is given by opposing the operator and complementing the probability value; we use this negation since we deal with attack and the relational expression was meant for goals not attacks.

A state $s = (S; M; E)$ is said to be an authentication attack state iff $\mathsf{EndAuth}(A, B, \mathrm{msg}) \in E \wedge \mathsf{beginAuth}(A, B, \mathrm{msg}) \notin E \wedge A \neq \mathsf{i}$ and $(\mathcal{P}_{(s)}^{trace}\ \neg\mathsf{comp})$. This violation means that both A and B do not agree on the message msg.

Note that since we deal with ground states we only iterate over states and do not quantify over agents or variables in terms.

V. Conclusion

The Dolev-Yao intruder model is widely used in formal verification for security protocols. On basis of this model, several tools enable the modelling of channel security as either totally insecure in which the intruder has full control on the whole network, or fully secure in which the intruder has very limited control. In this paper, we extended SPS language with probabilistic channels to model relatively secure channels in which the security of a channel is given by a probabilistic value. We define the semantics of the extended SPS by a translation to probabilistic automata via an intermediate translation step. The intermediate step is a translation to operational strands. Future work may include adding costs to channels to model the cost the intruder pays (in means of time, energy, effort, etc.) to break a secure channel. This can be done by attaching a cost to break secure channel to the probability of it. This allows checking the cost of an *attack* against certain values using certain model checking tools such as PRISM [20]. Such an extension is doable by means of probabilistic weighted automata as in [21] and [22]. Another future work may be the "strengthening" of the intruder; i.e., there are several attempts to add more strength to the intruder such as providing him with a certain probability an ability to break a cryptographic operation such as retrieving an encrypted message without having the key or getting the plain message of a hash as in [13]. Although [23] proved that it is equivalent to the classical Dolev-Yao intruder this can be an approach between abstract symbolic approach and computational approach.

References

[1] O. Almousa, S. Mödersheim, and L. Viganò, "Alice and Bob: Reconciling Formal Models and Implementation," 2015, programming languages with applications to biology and security - Colloquium in honour of Pierpaolo Degano for his 65th birthday, Revised Selected and Invited Papers.

[2] S. Mödersheim, "Algebraic Properties in Alice and Bob Notation," in *Availability, Reliability and Security, 2009. ARES'09. International Conference on*. IEEE, 2009, pp. 433–440.

[3] S. Mödersheim and L. Viganò, "The open-source fixed-point model checker for symbolic analysis of security protocols," *Foundations of Security Analysis and Design V*, pp. 166–194, 2009.

[4] A. Armando and L. Compagna, "Satmc: A sat-based model checker for security protocols," *Logics in Artificial Intelligence*, pp. 730–733, 2004.

[5] M. Turuani, "The cl-atse protocol analyser," *Term Rewriting and Applications*, pp. 277–286, 2006.

[6] AVISPA, "Deliverable D2.3: The Intermediate Format," 2003.

[7] B. Blanchet and B. Smyth, "ProVerif 1.85: Automatic Cryptographic Protocol Verifier, User Manual and Tutorial," 2011.

[8] M. Abadi and C. Fournet, "Mobile values, new names, and secure communication," in *POPL*, 2001, pp. 104–115.

[9] F. Fábrega, J. Herzog, and J. Guttman, "Strand spaces: Why is a security protocol correct?" in *Security and Privacy, 1998. Proceedings. 1998 IEEE Symposium on*. IEEE, 1998, pp. 160–171.

[10] O. Almousa, S. Mdersheim, and H. Nielson, "Security protocols: Specification, verification, implementation, and composition," Ph.D. dissertation, Technical University of Denmark (DTU), 2016.

[11] O. Almousa, S. Mödersheim, and L. Viganò, "Alice and Bob: Reconciling Formal Models and Implementation (Extended Version)."

[12] D. Dolev and A. Yao, "On the security of public key protocols," *Information Theory, IEEE Transactions on*, vol. 29, no. 2, pp. 198–208, 1983.

[13] R. Bresciani and A. Butterfield, "Weakening the dolev-yao model through probability," in *Proceedings of the 2Nd International Conference on Security of Information and Networks*, ser. SIN '09. New York, NY, USA: ACM, 2009, pp. 293–297. [Online]. Available: http://doi.acm.org/10.1145/1626195.1626265

[14] FutureID Project, "Deliverable D42.3: Future AnB: The projected APS Language of FutureID," 2013, www.futureid.eu.

[15] J. D. Guttman, "Establishing and preserving protocol security goals," *Journal of Computer Security*, vol. 22, no. 2, pp. 203–267, 2014. [Online]. Available: http://dx.doi.org/10.3233/JCS-140499

[16] O. Almousa, S. Mödersheim, P. Modesti, and L. Viganò, "Typing and compositionality for security protocols: A generalization to the geometric fragment," in *Computer Security - ESORICS 2015 - 20th European Symposium on Research in Computer Security, Vienna, Austria, September 21-25, 2015, Proceedings, Part II*, 2015, pp. 209–229.

[17] C. Weidenbach, D. Dimova, A. Fietzke, R. Kumar, M. Suda, and P. Wischnewski, "Spass version 3.5," in *Automated Deduction–CADE-22*. Springer, 2009, pp. 140–145.

[18] M. Stoelinga, "An introduction to probabilistic automata," *Bulletin of the EATCS*, vol. 78, pp. 176–198, 2002.

[19] R. Segala and N. Lynch, "Probabilistic simulations for probabilistic processes," in *CONCUR'94: Concurrency Theory*. Springer, 1994, pp. 481–496.

[20] M. Kwiatkowska, G. Norman, and D. Parker, "PRISM 4.0: Verification of probabilistic real-time systems," in *Proc. 23rd International Conference on Computer Aided Verification (CAV'11)*, ser. LNCS, G. Gopalakrishnan and S. Qadeer, Eds., vol. 6806. Springer, 2011, pp. 585–591.

[21] H. Hermanns and A. Turrini, "Cost preserving bisimulations for probabilistic automata," in *CONCUR 2013–Concurrency Theory*. Springer, 2013, pp. 349–363.

[22] K. Chatterjee, L. Doyen, and T. A. Henzinger, "Probabilistic weighted automata," in *CONCUR 2009-Concurrency Theory*. Springer, 2009, pp. 244–258.

[23] R. Zunino and P. Degano, "Weakening the perfect encryption assumption in dolev–yao adversaries," *Theoretical computer science*, vol. 340, no. 1, pp. 154–178, 2005.