A formal approach for automated reasoning about off-line and undetectable on-line guessing (short paper)

Bogdan Groza and Marius Minea

Politehnica University of Timişoara and Institute e-Austria Timişoara *
bogdan.groza@aut.upt.ro, marius@cs.upt.ro

Abstract. Starting from algebraic properties that enable guessing low-entropy secrets, we formalize guessing rules for symbolic verification. The rules are suited for both off-line and on-line guessing and can distinguish between them. We add our guessing rules as state transitions to protocol models that are input to model checking tools. With our proof-of-concept implementation we have automatically detected guessing attacks in several protocols. Some attacks are especially significant since they are undetectable by protocol participants, as they cause no abnormal protocol behavior, a case not previously addressed by automated techniques.

1 Motivation and related work

As password-based authentication continues to be used in practice and weak passwords are still chosen by users, detecting protocols subject to guessing attacks is a topic of high interest in security. In this paper we address the problem of formalizing a previously introduced approach to detect guessing attacks in a manner suitable for implementation in an automated verification toolset. We use IF (Intermediate Format), a specification language that can be handled by model checkers such as OFMC (Open Source Fixedpoint Model-Checker) [3] and SATMC (SAT-based Model Checker) [2] from the AVISPA toolset.

A previous intention of integrating guessing rules in OFMC exists in [9], which gives a formalization for off-line guessing attacks. In comparison, our contribution proposes a different formalism (our guessing rules are based on a different reasoning), which allows us to handle both on-line and off-line attacks. Our guessing rules are implemented at the level of the IF description language, without requiring the modification of the back-end model checkers. Other concrete implementations of guessing detection rules are by Corin et al. [7], Lowe [13] who used Casper/FDR and Blanchet [5] in ProVerif, a verifier based on Prolog rules. Our implementation is based on IF, a specification language which can be handled by several back-end model checkers, notably OFMC and SATMC, which thus gain the ability of detecting guessing attacks. Other theoretical foundations

* This work is supported in part by FP7-ICT-2007-1 project 216471, AVANTSSAR: Automated Validation of Trust and Security of Service-oriented Architectures

for reasoning on guessing attacks exist. Abadi et al. [1] use the indistinguishability of two terms, deduced by static equivalence, to formalize guessing. Equational theories for the applied pi-calculus are used by Corin et al. in [6], while Baudet [4] uses a constraint solving algorithm for an equational theory.

Our guessing rules are based on the pseudo-randomness properties of one-way functions. We consider two cases of guessing: first, when the adversary knows the image of a one-way function computed on the secret and other known input; second, when the adversary knows the image of a trapdoor function (encryption) with a key that depends on the secret, and can establish relations on its input. As one-way functions are pseudo-random, the output for a wrong secret cannot match any previously known value, thus a correct guess can be verified.

Most prior work addresses only off-line guessing, considering the low-entropy secret large enough to prevent guessing, or that unsuccessful attempts can be blocked. However, in some on-line attacks the protocol behavior is indistinguishable from normal. These attacks are undetectable by participants and especially dangerous. They are also realistic, as one of our case studies, a Norwegian ATM system, illustrates. Undetectable on-line attacks have also been the focus of Ding and Horster [8], but without a formalization or automated detection.

2 Formalization of guessing rules

To express the feasibility of guessing, we use the concept of *strongly distinguishing* functions [10], which cannot give the same output for two different secrets when these are paired with sufficiently many input choices.

Definition 1. Given $\sigma \in \{0,1\}^k$, we call a function $f(\sigma, x)$ *strongly distinguishing* in the first argument after q queries, if given any q distinct values $\{x_1, x_2, ..., x_q\}$, $\forall s_1 \neq s_2$ the probability that $f(s_1, x_i) = f(s_2, x_i)$ for all $i = 1 \,..\, q$ is at most 2^{-k}, i.e., $\forall s_1 \neq s_2 \,.\, Pr[f(s_1, x_i) = f(s_2, x_i), i = 1 \,..\, q] \leq 2^{-k}$.

Using strongly distinguishing functions, we have characterized the conditions for an intruder's guess in a *guessing lemma* [10]. However, due to its algebraic rather than symbolic formulation, it cannot be directly implemented in a formal verification tool. Therefore, we will link the concept of *strongly distinguishing* function (in one query) with a *symbolic* protocol description.

Protocol descriptions contain *terms*, which are either *atomic* or *composed*. *Atomic* terms are *variables*, *constants* or *numbers*; *composed* terms are formed by applying *pair*, *crypt*, *inv* and other predefined operators on *atomic* terms. *Facts* are predicates defined over terms, such as *iknows*, *state*, *contains*, etc.

Definition 2. We call a *symbolic protocol description* $\mathcal{P}$ a triple composed of an *initial state*, a set of *transition rules* and a set of *attack states*, i.e., $\mathcal{P} = (\mathit{InitialState}, \mathit{TransitionRule}^*, \mathit{AttackState}^*)$, where: i) the *initial state* is a conjunction of ground facts, ii) a *transition rule* has the form $\mathit{LHS} \Rightarrow \mathit{RHS}$ where *LHS* and *RHS* are conjunctions of facts, and *LHS* may also contain a negated fact and a *condition* (a conjunction of term equalities and inequalities), iii) an *attack state* is a conjunction of facts with a condition (like a *LHS*).

To reason about guessing, we define derivation rules $P \vdash_r T$, denoting that term T can be derived from term set P using rule r. Rule $\vdash_{ihears}$ *Term* denotes that a term has been overheard by the intruder during protocol execution.

Denote the set of symbols (constants or variables) appearing in *Term* by $\mathcal{S}(Term)$. If $s \in \mathcal{S}(Term)$ is such a symbol, we also write $Term \vdash_{part} s$.

Let $Term \vdash_{s \leftarrow gen(s')} Term'$ denote that $Term'$ is obtained by substituting any occurrence of the symbol s in *Term* with a fresh symbol $s' \notin \mathcal{S}(Term)$. For instance, $crypt(s, m) \vdash_{s \leftarrow gen(s')} crypt(s', m)$. As a particular case, we write $Term \vdash_{s \leftarrow igen(s')} Term'$ if s is substituted by a fresh value chosen by the intruder.

Consider a valuation function v defined on atomic terms with algebraic values, and extended to composed terms through function and operator application. We now relate our symbolic reasoning to the algebraic properties of the protocol.

Definition 3. A *symbolic protocol description* $\mathcal{P}$ is called algebraically dependent on symbol s, denoted $\mathcal{P}$ dep s if for any term *Term* such that $\vdash_{ihears} Term$ and $Term \vdash_{part} s$, and considering $Term'$ such that $Term \vdash_{s \leftarrow gen(s')} Term'$, for any valuation v such that $v(s) \neq v(s')$, we have $v(Term) \neq v(Term')$.

Given $s \in \mathcal{S}(Term)$, denote by $O_s^{Term}(\cdot)$ the oracle corresponding to the function obtained by making s a variable in *Term* and keeping other parts of it constant, e.g., $O_s^{crypt(s,m)}(\cdot)$ is the oracle corresponding to $f(s) = crypt(s, m)$.

Lemma 1. The *symbolic protocol description* $\mathcal{P}$ is *algebraically dependent* on s, i.e., $\mathcal{P}$ dep s, if and only if any function f obtained as $O_s^{Term}(\cdot)$ where $s \in \mathcal{S}(Term)$ and $\vdash_{ihears} Term$ is *strongly distinguishing* in one query.

Lemma 1 relates a *symbolic protocol description* with the algebraic notion of *strongly distinguishing* function in one query. Since injective functions are *strongly distinguishing* in one query, any symbolic protocol description in which a symbol s occurs only in the body of a injective (bijective) function is algebraically dependent on s. In practice, this covers a large class of protocols, since most cryptographic functions are bijective (even hash functions, if one assumes they are collision-free in the inputs that a protocol participant can provide).

Definition 4. An adversary *observes* an oracle for a secret s if it *hears* a term that contains s. The adversary *controls* an oracle for secret s if by replacing s in a term with a fresh s' (rule $\vdash_{s \leftarrow igen(s')}$) the adversary *knows* the new term.

$$\vdash_{ihears} Term \wedge Term \vdash_{part} s \Rightarrow observes(O_s^{Term}(\cdot)) \tag{1}$$

$$\vdash_{ihears} Term \wedge Term \vdash_{s \leftarrow igen(s')} Term' \wedge \vdash_{iknows} Term' \Rightarrow controls(O_s^{Term}(\cdot)) \tag{2}$$

Lemma 2. Consider a *symbolic protocol description* $\mathcal{P}$ such that $\mathcal{P}$ dep s. If an adversary *observes* and *controls* an oracle for a low-entropy secret s then the adversary can guess the secret s, i.e.,

$$observes(O_s^f(\cdot)) \wedge controls(O_s^f(\cdot)) \Rightarrow iguess(s) \tag{3}$$

The adversary can also guess if it *observes* messages encrypted with a key computed as a *strongly distinguishing* function on the secret, controls the corresponding decryption oracle, and can establish a relation to one or several parts of the encrypted messages. We formalize this case in what follows.

Definition 5. We call *s-dependent* an encryption or decryption oracle that uses a key containing s. An adversary that *hears* the encryption of some message with a key that contains s is said to *observe* an *s-dependent* encryption oracle. Moreover, we say that he *controls* the corresponding *s-dependent* decryption oracle if by replacing s in the encryption key with a fresh s' known to him the adversary can decrypt arbitrary messages encrypted with the new key, i.e.,

$$\vdash_{ihears} \{M\}_K \wedge K \vdash_{part} s \Rightarrow observes(O_s^{\{M\}_K}(\cdot)) \tag{4}$$

$$\{M\}_K \vdash_{\substack{s \leftarrow igen(s') \\ M \leftarrow gen(M')}} \{M'\}_{K'} \wedge \vdash_{iknows} M' \Rightarrow controls(O_s^{\{M\}_{K^{-1}}}(\cdot)) \tag{5}$$

Here, $\{M\}_K$ is the encryption of message M with key K. To keep relation (5) simple, we've left implicit that the adversary must overhear the term $\{M\}_K$ and the encryption key must contain s, i.e., $\vdash_{ihears} \{M\}_K \wedge K \vdash_{part} s$ as a premise. This is of course needed for the question of controlling the oracle to make sense.

To express a relation between encrypted inputs we first need a derivation rule to produce all distinct messages M that satisfy a property $Fact(M)$, by concatenating them into term T, denoted $Fact \vdash^M_{concat} T$. For example, $(\vdash_{ihears} M) \vdash^M_{concat} T$ yields a term T that is the concatenation of all distinct terms for which $\vdash_{ihears} M$ holds. Similarly, $(\vdash_{ihears} \{M\}_K \wedge K \vdash_{part} s) \vdash^M_{concat} T$ produces the concatenation of all distinct messages that are encrypted with a key that contains s. Also, let $T \vdash_{split} \langle T'.T'' \rangle$ denote that T' and T'' are derived by splitting T into disjoint subsets of terms (at least one of them non-empty).

The second guessing rule requires powerful capabilities: to find a relation between two terms (the *relates* fact) the adversary can use any available operators: *pair*, *crypt*, etc., as well as his Dolev-Yao abilities, *fake*, *overhear*, etc. Thus, for deciding *relates* the adversary can perform any transition allowed by the *symbolic protocol description* $\mathcal{P}$. The following definition models this intuition.

Definition 6. An adversary can *relate* two terms T' and T'' of a *symbolic protocol description* $\mathcal{P}$ if by adding T' to the adversary knowledge he can derive T'' (denoted $T' \vdash_{DY(\mathcal{P})} T''$) using *all* its abilities over $\mathcal{P}$.

$$T' \vdash_{DY(\mathcal{P})} T'' \Rightarrow relates(T', T'') \tag{6}$$

Lemma 3. Let $\mathcal{P}$ be a *symbolic protocol description* such that $\mathcal{P}$ dep s. If the adversary *observes* one or more *s-dependent* encryption oracles for which he *controls* the corresponding decryption oracles and can *relate* parts of the encrypted messages then the adversary can guess the secret, i.e.,

$$\begin{aligned} observes(O_s^{\{M\}_K}(\cdot)) \ \wedge controls(O_s^{\{M\}_{K^{-1}}}(\cdot)) \vdash^M_{concat} T \\ \wedge\ T \vdash_{split} \langle T'.T'' \rangle \wedge\ relates(T', T'') \Rightarrow guess(s) \end{aligned} \tag{7}$$

3 Implementation and experimental results

Our formalization of the guessing calculus makes it amenable to an implementation where where states are sets of terms, and transitions are given as rewrite rules, as in the IF protocol specification language. Derivations such as $\vdash_{ihears}$,

$\vdash_{part}$, $\vdash_{s \leftarrow gen(s')}$, $\vdash_{split}$ yield corresponding IF facts. These are combined into rules to establish the relations *observes* and *controls*, and ultimately, guessing.

We use an adversary model with standard Dolev-Yao abilities: the adversary can fake new messages, intercept sent messages or overhear them. Moreover, the adversary has the standard computational abilities: he can encrypt and decrypt if he knows the corresponding key, and he can pair and decompose messages.

Based on this model we want to express rules for the adversary's ability to *observe* and *control* oracles. To decide whether a composed term represents an oracle, we need to determine if it contains the secret to be guessed. By overhearing such a term, the adversary *observes* the oracle. Further, to decide *controls*, we start from terms containing the secret, construct new terms in which the secret is replaced by a different value and test if the adversary knows them, and thus *controls* the oracle for the function derived from the term.

For secret containment (the derivation $\vdash_{part}$ in our theory) we define the `containsSec` fact, which is true for all terms containing the secret. For secret replacement (derivation $\vdash_{s \leftarrow gen(s')}$), we define the `replaceSec` fact which replaces any secret from the `guessableSecrets` set with a *replacement* secret.

With these helper facts defined, the *observes* and *controls* abilities are easily derived. *Observing* an oracle is modeled as `ihears(T).containsSec(T, SList)`, where `SList` is the list of guessable secrets, while *controlling* an oracle is specified as `replaceSec(T, Tnew).iknows(TNew)` (where pairing with `.` means fact conjunction in IF). Explicit *observes* and *controls* predicates are not necessary; for efficiency, the above expressions are directly embedded into the guessing rules.

Guessing multiple secrets. To enable guessing in such scenarios, secrets already guessed must be used in subsequent guesses. However, this cannot be expressed by a simple chaining of the guesses, since adding new knowledge to the intruder cannot be done dynamically in the attack condition. Our simple and effective solution expresses the guessing rule (based on the *observes* and *controls* abilities) as transition of the protocol itself. As a result, any guessed value is added to *iknows*. Being protocol-independent, this rule can be inserted in any protocol specification and enables chaining multiple guesses.

Distinguishing detectable from undetectable on-line attacks. As a first intuition, if guessing takes place after a participant reached a final state, then guessing goes undetected for that participant. This intuition is wrong, as the same participant may have another instance still running. To distinguish undetectable from detectable on-line guessing attacks, we need to express that all participant instances have successfully completed. We can do this by adding the PIDs of all started instances to a set, adding their termination to the intruder knowledge and checking the match in the attack condition. Alternatively, simply matching the count of started and finished instances suffices.

MS-CHAP and NTLM. These are two simple, well known protocols from Microsoft, vulnerable but still frequent in practice even today. MS-CHAP is used for remote user authentication and has two versions. NTLM is used with SMB to access remote printers, files etc. and has three versions: NTLMv1, NTLMv2 and NTLMv2-Session. Figure 1 presents MS-CHAP v2 and NTLM v2-Session.

MS-CHAP v2

1. $A \rightarrow B : A$
2. $B \rightarrow A : N_B$
3. $A \rightarrow B : N_A, H(k_{AB}, N_A, N_B, A)$
4. $B \rightarrow A : H(k_{AB}, N_A)$

NTLMv2-Session

1. $B \rightarrow A : N_B$
2. $A \rightarrow B : N_A, H(k_{AB}), H'(N_A, N_B))$
3. $B \rightarrow A : H(k_{AB}, H'(N_A, N_B)), H'(N_A, N_B)$

(a,1) → i: a
i → (b,1): a
(b,1) → i: Nb(2)
i → (a,1): Nb(2)
(a,1) → i: Na(3).h(kab.Na(3).Nb(2).a)
i → (b,1): Na(3).h(kab.Na(3).Nb(2).a)
(b,1) → i: h(kab.Na(3))
i → (a,1): h(kab.Na(3))
i → (i,1): h(kab_rpl.Na(3))
i → (i,1): kab.snull
i → (i,17): kab

Fig. 1. MS-CHAP and NTLMv2-Session protocols and OFMC attack trace

We have augmented the MS-CHAP v2 protocol model with guessing rules. As expected, OFMC found the attack in Figure 1; a similar attack can be traced for NTLM. The intruder acts as man-in-the-middle. Guessing is possible because the intruder hears $h(kab.Na(3))$, and knowing $Na(3)$ can compute $h(kab\_rpl.Na(3))$ for arbitrary replacements kab_rpl of kab. By Lemma 2, this means that the intruder *observes* and *controls* the oracle $O_s^f(\cdot)$, where $f = h(s, Na(3))$. The last three trace steps are intruder reasoning; they reflect the fact that additions to intruder knowledge are modeled in the same way as message receipts.

The guessing attacks on MS-CHAP and NTLM are known and simple, but the results serve as basic proof that our approach can automate their detection. The role of an automatic verification tool is to be used on large, complex systems or services that cannot be handled by hand and where such a protocol is only a small foundational component. Thus, the ability of detecting flaws in such a protocol becomes crucial in the discovery of new flaws in the overall system.

The Norwegian ATM. A second test for our implementation was a Norwegian ATM protocol (Fig. 2), known to be flawed [11]. A bank issues ATM cards with the user PIN encrypted as $E_{BK}(PIN)$, where BK is a secret bank key. The question is if an adversary, having a stolen card, can guess the user PIN from the encrypted value. One can argue that testing on-line against an ATM with all possible PINs is not feasible since the ATM will lock after the first, say 3 wrong guesses. However, imagine that the adversary has a card issued by the same bank, with $E_{BK}(PIN_{adv})$ on it and that each card owner may change its PIN at an ATM. Now, the adversary can break the PIN on the stolen card by using an on-line attack in which he legally changes its own PIN and then verifies the encrypted value from his card against the one from the stolen card. This is a simplified example, since the PIN is usually not encrypted alone, but with some card-specific information. However, it justifies the concern for on-line attacks.

The first attack trace produced by OFMC (Fig. 2) was rather unexpected. If an adversary obtains $DES_{BKey}(PIN)$, he can compute $DES_k(m)$ for any key and message, thus he *observes* and *controls* the oracle $O^{DES_{(\cdot)}(\cdot)}$ and can perform guessing. This is an off-line attack and the trace represents intruder deductions: controlling the oracle with replaced values (1), the intruder deduces both PIN

Card Issuing Stage: 1. $Bank \rightarrow User : \lfloor DES_{BKey}(PIN) \rfloor_{16}, PIN$

PIN Change Procedure: 1. $User \rightarrow ATM : \lfloor DES_{BKey}(PIN_{old}) \rfloor_{16}, PIN_{old}, PIN_{new}$

2. $ATM \rightarrow User : \lfloor DES_{BKey}(PIN_{new}) \rfloor_{16}$

First attack trace	Second attack trace
1. i → (i,3): $\text{pinStolen\_rpl}_{\text{BKey\_rpl}}$	1. i → (i,1): $\text{PIN(1)}_{\text{BKey}}$.PIN(1)
2. i → (i,3): BKey.pinStolen	2. i → (i,2): $\text{PIN(1)}_{\text{BKey}}$.PIN(1).$\text{pinStolen}_{\text{BKey}}$
3. i → (i,17): pinStolen	3. i → (i,2): pinStolen

Fig. 2. The Norwegian ATM protocol (modified) and OFMC attack traces

and *BKey* (2), and thus the PIN (3). In practice this is impossible because every PIN would match for some DES key; moreover, only 16 bits of the result are stored, yielding a huge number of potential values for *BKey* and PIN. To test our calculus, we restricted the adversary from trying replacements for *BKey*. Thus, the adversary no longer controls the oracle $O^{DES_{(\cdot)}(\cdot)}$.

OFMC found the second trace, where the adversary, being issued a legal card (1) uses the PIN change procedure against the ATM in order to *control* the oracle $O^{DES_{BKey}(\cdot)}$, where *BKey* is constant. Matching the terms for legal and stolen cards in the intruder knowledge (2) produces the PIN (3). The attack is realistic assuming that the DES encryption is done directly on the PIN.

The Lomas et al. protocol [12] is an interesting case study for guessing attacks. The protocol is illustrated in Figure 3. With respect to our theory it is relevant as it fits the second guessing case (Lemma 3). Lowe found an attack by choosing the constant 0 as timestamp, which allows a replay attack that lets the adversary recover the nonce from two different responses of the server, thus allowing the verification of a correct password [13]. Using OFMC we found a different attack, based again on the weakness of the timestamp. We used a nonce encrypted with *pwdA* as the arbitrary timestamp, in order to avoid Lowe's attack. Quite unexpectedly, OFMC produced the following attack trace: instead of replaying the message from step 1 (Lowe's attack) the intruder lets the protocol run normally between A and B and from this run he obtains $\{Na1, k \oplus Na2\}_{pwdA}$. Now the intruder initiates a new protocol session, impersonating A and sending $\{A, B, Na1', Na2', Ca, \{Na1, k \oplus Na2\}_{pwdA}\}_{pks}$ in step 1 to the server. Indeed $\{Na1, k \oplus Na2\}_{pwdA}$ besides the length (which is mainly an implementation issue) is just a random value (indistinguishable from a nonce) encrypted with the correct password of A. Further on, the server answers in step 4 with $\{Na1', k \oplus Na2'\}_{pwdA}$, but now $Na1'$ is known to the adversary as he has forged the message in step 1 and thus he can make a correct guess. This attack is not based on a replay, as the message in step 1 was never received by S before. To the best of our knowledge, this attack is new.

1. $A \rightarrow S : \{A, B, Na1, Na2, Ca, \{Ta\}_{pwdA}\}_{pks}$	5. $S \rightarrow B : \{Nb1, k \oplus Nb2\}_{pwdB}$
2. $S \rightarrow B : A, B$	6. $B \rightarrow A : \{Rb\}_k$
3. $B \rightarrow S : \{B, A, Nb1, Nb2, Cb, \{Tb\}_{pwdB}\}_{pks}$	7. $A \rightarrow B : \{f(Rb), Ra\}_k$
4. $S \rightarrow A : \{Na1, k \oplus Na2\}_{pwdA}$	8. $B \rightarrow A : \{f(Ra)\}_k$

Fig. 3. The Lomas et al. protocol

4 Conclusions

We have formalized rules for detecting guessing attacks, linking their underlying algebraic properties to the context of symbolic protocol descriptions. Stated as conditional rewrite rules in the description language IF, they can be added to any protocol model, and used with the usual Dolev-Yao intruder deductions. Thus, guessing attacks can be automatically detected without change to the model-checker back-ends. Our implementation automatically distinguishes between detectable and undetectable on-line attacks and can guess multiple secrets.

Using OFMC we have found attacks on several protocols; of these, the attacks on the simplified Norwegian ATM (using the PIN change procedure) and on the Lomas et al. protocol are new to the best of our knowledge. We believe this shows our automation of guessing attack detection to be practically relevant, especially in its support for undetectable on-line attacks. As future work we intend to integrate this proof-of-concept implementation into the AVANTSSAR verification toolset or potentially with a high-level specification language.

Acknowledgments. We thank Sebastian Mödersheim and Roberto Carbone for valuable answers on using the OFMC and SATMC model checkers.

References

1. M. Abadi, M. Baudet, and B. Warinschi. Guessing attacks and the computational soundness of static equivalence. In *Proc. FoSSaCS 2006*, *LNCS* 3921, pp. 398–412.
2. A. Armando and L. Compagna. SAT-based Model-Checking for Security Protocols Analysis. *Internat. Journal of Information Security*. 21 September 2007.
3. D. Basin, S. Mödersheim, and L. Viganò. OFMC: A symbolic model checker for security protocols. *Internat. Journal of Information Security*, 4(3):181–208, 2005.
4. M. Baudet. Deciding security of protocols against off-line guessing attacks. *Proc.* 12^{th} *ACM Conf. on Computer and Communications Security*, pp. 16–25, 2005.
5. B. Blanchet. An Efficient Cryptographic Protocol Verifier Based on Prolog Rules. In 14^{th} *IEEE Computer Security Foundations Workshop*, pp. 82–96, 2001.
6. R. Corin, J. M. Doumen, and S. Etalle. Analysing password protocol security against off-line dictionary attacks. In *Proc. WISP 2004*, pp. 47–63.
7. R. Corin, S. Malladi, J. Alves-Foss, and S. Etalle. Guess what? here is a new tool that finds some new guessing attacks. In *Proc. WITS 2003*, pp. 62–71.
8. Y. Ding and P. Horster. Undetectable on-line password guessing attacks. *Operating Systems Review*, 29(4):77–86, 1995.
9. P. H. Drielsma, S. Mödersheim, and L. Viganò. A formalization of off-line guessing for security protocol analysis. In *Proc. LPAR 2004*, *LNCS* 3452, pp. 363–379.
10. B. Groza, M. Minea. A calculus to detect guessing attacks. In *Proceedings of the* 12^{th} *Information Security Conference*, *LNCS* 5735, pp. 59–67, Springer, 2009.
11. K. J. Hole, V. Moen, A. N. Klingsheim, and K. M. Tande. Lessons from the Norwegian ATM system. *IEEE Security and Privacy*, 5(6):25–31, 2007.
12. T. Lomas, L. Gong, J. Saltzer, R. Needhamn. Reducing risks from poorly chosen keys. *Proc. of* 12^{th} *ACM Symp. on Operating Systems Principles*, pp. 14–18, 1989.
13. G. Lowe. Analysing protocols subject to guessing attacks. *Journal of Computer Security*, 12(1):83–98, 2004.