Automated algebraic analysis of structure-preserving signature schemes

Joeri de Ruiter
Institute for Computing and Information Sciences
Radboud University Nijmegen
P.O. Box 9010, 6500 GL Nijmegen, The Netherlands
joeri@cs.ru.nl *

Abstract

Structure-preserving signature schemes can be very useful in the construction of new cryptographic operations like blind signatures. Recently several of these schemes have been proposed. The security of signature-preserving signature schemes is still proved by hand, which can be a laborious task. One of the ways to prove security of these schemes algebraic analysis can be used. We present an approach to perform this analysis and the first tool, *CheckSPS*, that can do an algebraic security analysis of these schemes, using SMT solvers as backend. This can help in constructing new schemes and analyse existing schemes. Our tool can handle all the common security objectives for signature schemes, i.e. existential unforgeability and strong existential unforgeability, and all the common capabilities for adversaries, i.e. random message attacks, non-adaptive chosen message attacks and adaptive chosen message attacks. The tool is sound, so if an attack is found it is actually possible to construct a forged signature.

1 Introduction

Digital signatures are widely used in today's information systems. When combining digital signatures with zero-knowledge proofs we can construct powerful new operations like, for example, blind signatures, where a proof is given that a ciphertext contains a signature over a given message without revealing the actual signature. To combine a signature scheme and zero-knowledge proofs, structure-preserving signature schemes are particularly useful. Structure-preserving signature (SPS) schemes use group elements for all elements of the messages and signatures. The groups that are used are bilinear groups and the operations used for signing are generic group operations. Verification of signatures is done using bilinear pairing equations. These kind of schemes are perfect to be used with Groth-Sahai non-interactive zero-knowledge (NIZK) proofs, because these proofs are efficient and usable for equations using bilinear pairings [1].

To prove security of SPS schemes there are two approaches that can be used: an algebraic analysis and a reduction to a hard problem. Where with algebraic analysis we make assumptions on the messages the attacker can construct, with reduction based proofs we assume the attacker cannot efficiently solve one or more hard problems. Until now SPS schemes are usually constructed based on a particular hard problem and proven by hand, a very laborious task. We discuss an approach to perform an algebraic analysis and introduce the first tool, *CheckSPS*, that can aid in these proofs by automating the algebraic analysis of SPS schemes. The tool is sound, meaning all attacks that are found lead to actual forgeries. Completeness can be provided, for a given number of oracle queries, depending on the mode that is used as we will discuss is Section 4. Ultimately, automation of the proofs could be used to automatically synthesise SPS schemes following the approach by Barthe et al. presented in [2] for padding-based encryption schemes.

*Most of this work was carried out while visiting Microsoft Research in Cambridge for an internship

Scheme	**Type**	$\|\boldsymbol{message}\|$	$\|\boldsymbol{vk}\|$	$\|\boldsymbol{signature}\|$	$\|\mathcal{V}\|$	**Assumptions**
[4]	1	n	$\mathcal{O}(n)$	$\mathcal{O}(n)$	Many	DLIN
[5]	1	2	1	5	3	q-DHSDW, WFCDH
[5]	3	1	$(1,1)$	$(3,2)$	3	q-ADHSDW, AWFCDH
[7]	3	$(n,0)$	$(4,2n+8)$	$(5,2)$	2	q-SFP
[8]	3	$(n,0)$	$(1,n+4)$	$(3,1)$	2	q-type
[9]	3	$(n,0)$	$(7,n+13)$	$(7,4)$	4	SXDH, XDLIN1
[7]	3	(n,m)	$(m+5,n+12)$	$(10,3)$	3	q-SFP
[8]	3	(n,m)	$(m+3,n+4)$	$(3,3)$	2	q-type
[9]	3	(n,m)	$(m+8,n+14)$	$(8,6)$	5	SXDH, XDLIN1
[14]	3	$(1,0)$	$(0,1)$	$(2,1)$	2	q-type

Table 1: Overview of existing SPS schemes. $|message|$, $|vk|$ and $|signature|$ are the number of group elements in the messages, verification keys and signatures respectively. $|\mathcal{V}|$ is the number of verification equations. For schemes using Type 3 bilinear groups, (n,m) indicates the number of elements from the different groups (i.e. n group elements from $\mathbb{G}_1$ and m from $\mathbb{G}_2$).

2 Structure-preserving signature schemes

An structure-preserving signature scheme consists of system parameters ($\mathcal{SP}$) and sets of all plaintext messages ($\mathcal{M}$), signatures ($\mathcal{S}$), signing keys ($\mathcal{SK}$), verification keys ($\mathcal{VK}$), a signing algorithm (*sign*) and verification equations ($\mathcal{V}$) used in a verification algorithm (*verify*).

The schemes make use of bilinear pairings. Following [3], we can distinguish between different types of bilinear pairings. In this paper we will consider Type 3 bilinear pairings. For these there are three finite cyclic groups ($\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_t$) of order q and a bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_t$ such that $e(g_1^a, g_2^b) = g_t^{ab}$ and there are no efficiently computable isomorphisms between groups $\mathbb{G}_1$ and $\mathbb{G}_2$. If $\mathbb{G}_1$ and $\mathbb{G}_2$ are equal, this would be considered a Type 1 bilinear pairing. We use g_1, g_2 and g_t as generators for the groups $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_t$ respectively.

The verification equations in SPS schemes are pairing product equations:

$$\prod_i \prod_j e(A_i, B_j)^{c_{i,j}} = Z,$$

where A_i and B_j are terms from the system parameters, signature, message or verification key. The terms from the system parameters and verification key are constants in the verification process. The ones from the signature and message are variables, as these are provided as external input to the verifier.

The first SPS scheme was introduced in [4]. This scheme is not practical though as it requires thousands of group elements for the signatures. The term 'structure-preserving signatures' was only introduced in later publications. After this first work, more efficient schemes were introduced and different security assumptions used [5–14]. Structure-preserving signature schemes can be characterised by the number of elements in the messages, verification keys, signatures and the number of verification equations. In Table 1 we give an overview of various schemes, that all were proved secure by hand. For the definitions of the different security assumptions we refer the reader to the corresponding papers, as we will not discuss these in detail here.

3 Algebraic analysis

To prove a signature scheme secure, we want to proof that an adversary can construct a valid signature without knowing the secret key only with negligible probability. Here we distinguish

between two security objectives:

- **EUF (Existential Unforgeability)**: the adversary cannot construct a valid signature on a message for which he hasn't seen a signature yet.
- **sEUF (Strong Existential Unforgeability)**: the adversary cannot construct a valid signature for any message. This includes constructing new signatures for messages he has already seen a signature for.

In general the second notion would be preferred. However, EUF is, for example, useful for schemes allowing re-randomisation of signatures. For these schemes it is by definition possible to construct a new signature by randomising an old one and sEUF would thus never hold. Next to the security objective we want to proof, we need to define the capabilities of the adversary. Traditionally, when proving security of signature schemes the adversary is assumed to have access to a signing oracle. The signing oracle can be used to obtain signatures on messages. The messages that the adversary can request signatures for from the oracle define how powerful the attacker is. We consider three different models for the adversary:

- **RMA (Random Message Attack)**: the adversary is given access to an oracle that outputs a signature/message pair for a random message. The adversary cannot influence the messages he gets the signatures for. This is the weakest form of a signing oracle we consider.
- **NA-CMA (Non-Adaptive Chosen Message Attack)**: the adversary is given access to an oracle that outputs a signature for given messages. He has to decide on the messages to be signed on beforehand and receives the signatures only after picking the messages. The messages are thus independent from the signatures but the different messages could depend on each other.
- **CMA (Adaptive Chosen Message Attack)**: the adversary is given access to an oracle that outputs a signature for a given message, which might depend on previous messages or signatures. This oracle is the most powerful the adversary can be given access to.

Next we will discuss an approach to perform an algebraic security analysis of structure-preserving signatures schemes. In the algebraic analysis we assume the adversary tries to construct a forgery by combining known terms like elements from, for example, signatures, messages or the verification key. This is by definition interactive and relies on a q-type assumption, where the security bound depends on the number of oracle queries made by the adversary, in this case the number of signatures he requests.

To decide whether the adversary can construct a valid message/signature pair using his knowledge we need to represent all the possible terms he can construct. Assuming his initial knowledge contains the list $T_1 \leftarrow \mathbb{G}_1^n$ of n terms from group $\mathbb{G}_1$, the terms in $\mathbb{G}_1$ that can be constructed are given by $\bigcup_{C \in \mathbb{Z}_q^{|T|}} \prod_{t \in T_1} \prod_{0 \leq i < |C|} t^{C_i}$. The same can be done to determine the possible knowledge of terms from $\mathbb{G}_2$. As all terms in the messages and signatures are either an element in $\mathbb{G}_1$ or $\mathbb{G}_2$ we do not need to consider the construction of knowledge of terms from $\mathbb{G}_T$. In the analysis of SPS schemes, the adversary has an initial knowledge $\mathcal{K}_0$ consisting of the public parameters of the SPS scheme and possible messages he is given or can construct.

When considering RMA security, the adversary does not have any influence on the messages that are being signed. Therefore, as messages that the attacker obtains from the oracle we simply take random group elements. The initial knowledge is as follows

$$\mathcal{K}_0 = \mathcal{SP} \cup \mathcal{VK} \cup \mathcal{M},$$

where $\mathcal{M}$ is a set of random messages. The size of $\mathcal{M}$ is equal to the number of oracle queries the adversary is given. The final knowledge of the adversary, that he can use to construct forgeries, is the following in this case

$$\mathcal{K} = \mathcal{K}_0 \cup \bigcup_{m \in \mathcal{M}} \mathit{sign}(\mathcal{SK}, m).$$

For non-adaptive CMA security the messages are chosen by the adversary and can thus depend on terms from the system parameters $\mathcal{SP}$ and the verification key $\mathcal{VK}$. The knowledge is constructed as with RMA, however, the initial knowledge is now defined as follows:

$$\mathcal{K}_0 = \bigcup_{i\in\{1,2\}} \bigcup_{t\in\mathcal{P}(\mathcal{SP}_i\cup\mathcal{VK}_i)} \prod_{a\in t} a,$$

where $\mathcal{SP}_i$ and $\mathcal{VK}_i$ are all terms in $\mathcal{SP}$ and $\mathcal{VK}$ in $\mathbb{G}_i$. $\mathcal{K}_0$ thus contains all possible combinations of elements from the system parameters and verification key. The final knowledge is now defined as

$$\mathcal{K} = \mathcal{K}_0 \cup \bigcup_{m\in\mathcal{K}_0} sign(\mathcal{SK}, m).$$

$\mathcal{SP}$ and $\mathcal{VK}$ are no longer explicitly included as they are already part of $\mathcal{K}_0$.

As with adaptive CMA, new messages can depend on previous messages or signatures, so we get a recursive definition for the knowledge in this case. The initial knowledge of the adversary is the same as for non-adaptive CMA:

$$\mathcal{K}_0 = \bigcup_{i\in\{1,2\}} \bigcup_{t\in\mathcal{P}(\mathcal{SP}_i\cup\mathcal{VK}_i)} \prod_{a\in t} a$$

Now the adversary's knowledge after n oracle queries is

$$\mathcal{K}_n = \bigcup_{i\in\{1,2\}} \bigcup_{t\in\mathcal{P}(\mathcal{K}_{n-1,i}\cup\{sign(SK,m)\})} \prod_{a\in t} a,$$

where $m \in \mathcal{K}_{n-1}$ and $n > 0$.

As we only check one instance of the scheme at a time, we will include the system parameters in the verification key.

4 Automated analysis

To check the security of an SPS scheme we want to decide whether it is possible for an adversary to come up with terms, constructed from his knowledge, that satisfy the verification equations, i.e., whether he can come up with a forged signature. In the analysis we replace every variable in the verification equations by the product of terms from the adversary's knowledge that correspond to the same group. We then try to find coefficients such that the equations hold. If we can find these coefficients, it means there is an attack against the scheme. The message/signature pairs that are requested by the adversary from the oracle are discarded as these are trivially correct solutions. If we want to check for strong unforgeability (sEUF), we also discard all solutions that are exactly the same as the given message/signature pairs. A solution with a new signature for an old message is thus considered a valid attack in this case. For regular unforgeability (EUF), we exclude all solutions with messages for which the signature was requested by the adversary. For an attack it is thus necessary in this case to come up with a valid signature for a message that was not signed before.

For the analysis all the verification equations are rewritten to normal form. For this we make use of the static equivalence properties for bilinear pairings as shown in [15]. This is done by representing all terms by multivariate polynomials and combining these polynomials. If we only consider constant terms and we, for example, have the term $g_1^a g_1^{bc}$, this will be represented by the polynomial $a + bc$. Now the equation $e(g_1^a, g_2^b g_2^c)^d e(g_1, g_2^e) = g_t^f$ is rewritten to the equation $a * (b + c) * d + e = f$, making use of the property of bilinear pairings as given in Section 2. Next we consider the case where the knowledge of the adversary consists of the two terms g_1^a and g_1^b. Assume we have the verification equation $e(v, g_2^c)e(w, g_2) = g_T^d$, where v and w are variables. This equation results in the polynomial equation $(x_0 * a + x_1 * b) * c + (y_0 * a + y_1 * b) = d$. We would now look for values for the coefficients x_0, x_1, y_0 and y_1 that satisfy the equation to find an attack. To

do this, first, the polynomial is converted to normal form by simplifying it and ordering it based on lexicographic ordering of the variables: $y_0 * a + y_1 * b + x_0 * ac + x_1 * bc = d$. By rewriting all the verification equations to normal form we end up with two polynomials that need to be equal for each equation. If we have one verification equation for two polynomials, for every monomial the coefficients have to be equal on both sides of the equation. This results in an equation system with an equation for every coefficient on either side of the equation. If we would have the following example $x_0 * a + x_1 * b + x_2 * cd = y_0 a + y_1 * c + y_2 * cd$, this would give the equation system $x_0 = y_0$, $x_! = 0$, $x_2 = y_2$, $y_1 = 0$. A solution to these equation equals a valid signature.

Our tool *CheckSPS* performs the algebraic analysis described above for a fixed number of oracle queries. The tool is written in OCaml and is about 2000 lines of code. It makes use of Why3, which functions as an intermediate layer between our tool and several SMT solvers, such as Z3, Alt-Ergo and CVC3 [16–19]. Using Why3 has the advantage that the user can use any solver, including future ones, he prefers as long as it is supported by Why3. The tool can perform a sanity check on the verification equations to decide whether they are valid. For the actual analysis, the tool supports the different adversary capabilities (RMA, non-adaptive CMA and adaptive CMA) and security objectives (EUF and sEUF) for a fixed number of oracle queries. The tool first reduces an SPS scheme to normal form and constructs the equation system corresponding to the adversary's capabilities and the security objective. This equation system is then passed to Why3, which in turn gives it to an SMT solver. The solver that is to be used can be given to *CheckSPS* as an argument.

The tool has two different modes. The first mode tries to find solutions for the equation system in $\mathbb{Z}$ instead of $\mathbb{Z}_q$. As we do not have the inverse operation in $\mathbb{Z}$, a consequence of this is that the tool is not complete in this mode. Solutions where two coefficients are the inverse of each other might thus be missed. The second mode uses a type for generic fields provided by Why3. Though complete, this mode is unfortunately very slow or solvers do not succeed in solving the equations at all, usually because the system runs out of memory.

5 Performance

We will now discuss the performance of the tool on two schemes proposed by Abe et al. in [8]. Below we give the input to our tool for the minimal scheme presented in Section 4 of [8]:

```
vk {
  g = g1;
  h = g2;
  gu = g1^u;
  hv = g2^v;
  hw = g2^w;
  hz = g2^z
}

sk {
  u: Zq;
  v: Zq;
  w: Zq;
  z: Zq
}

message {
  gm: G1;
  hn: G2
}

signature {
```

#queries	EUF	sEUF
2	0.23s	0.26s
3	0.38s	0.52s
4	1.19s	1.07s
5	3.80s	2.32s
6	9.06s	5.08
7	21.01s	10.34s
8	60.39s	20.38s
9	82.86s	31.97s
10	175.24s	69.49s

Table 2: Timings for different number of oracle queries with RMA for minimal SPS scheme from [8] (average over 10 runs)

```
  gr: G1 = g^r;
  gs: G1 = g^(z-(r*v)) * gm^-w;
  ht: G2 = (h * hn^-u)^(1/r)
}

equations {
  e(g, hz)  = e(gr, hv) * e(gs, h) * e(gm, hw);
  e(g, h) = e(gr, ht) * e(gu, hn)
}
```

The tests were performed on a standard laptop with an Intel Core i5 M540 CPU and 4GB RAM. We picked Z3 as solver to be used by the tool and used the mode for solving the equations in $\mathbb{Z}$. We start by considering RMA analysis for up to 10 oracle queries for the scheme from Section 4 in [8]. The results can be found in Table 2. Though the running time seems to increase exponentially, for a low number of queries the tool is still very fast. As the number of queries increases, the analysis for sEUF becomes faster than the one for EUF. This is probably due to the fact that with sEUF there are more side conditions that the solvers can use to optimise the computations. With sEUF there are restrictions that previous message/signature pairs cannot be used as a valid attack, whereas with EUF only previous messages are considered to be invalid solutions. In Table 3 we can see the difference between the analysis for the different capabilities. The polynomials representing the knowledge of the adversary become increasingly complex for RMA, NA-CMA and CMA. This is clearly visible in the table and the EUF analysis for CMA could not finish for this scheme within 24 hours. However, for sEUF the timings are still very reasonable. This analysis provides stronger security guarantees than EUF and is therefore more useful in most cases.

In Section 5.3 in [8], a re-randomisable scheme is presented. When analysing this scheme we can see the difference between EUF and sEUF analysis. For the EUF no attack is found but sEUF the tool does find an attack. This is as expected as due to the re-randomisation any given signature can be transformed in a new signature on a previously signed message.

The current version of the tool does not yet return an actual attack if one is found. This is because Why3 does not return counterexamples from the underlying solvers. However, using output from, for example, Z3 directly this could be possible.

	EUF	**sEUF**
2-RMA	0.21s	0.25s
2-NA-CMA	9.33s	2.06s
2-CMA	N/A	24.18s

Table 3: Timings for minimal structure-preserving signatures from [8] (average over 10 runs)

6 Conclusion

We presented a method for algebraic analysis of structure-preserving signature schemes and the tool *CheckSPS*, that can be used in the security analysis of these schemes for different security objectives (EUF and sEUF) and using different capabilities for the adversary (RMA, NA-CMA and CMA). In the same way that protocol verification tools such as ProVerif provides valuable support in finding security flaws in security protocols, we now have the very first tool that can find flaws in SPS schemes [20]. The tool can easily cope with schemes that are proposed in the literature and can provide results in a less than a second to several minutes for more complicated analysis. Even support is provided for a mode for complete analysis for a given number of oracle queries, though in practice this is a challenge for SMT solvers. In future work we would like to extend the tool such that attacks are automatically returned in a readable form. Also, it is interesting to find bounds for SPS schemes that determine the maximum number of oracle queries for which a scheme has to be proved correct for it to be secure for an unbounded number of queries. Another direction of research is to look at automating proofs based on hardness assumptions instead of only algebraic analysis. This is not as deterministic as the analysis discussed in this paper and is actually an NP hard problem. Ultimately these automated proving techniques could be used to automatically synthesise structure-preserving signature schemes and find efficient minimal schemes.

Acknowledgements

This work was done while visiting Microsoft Research in Cambridge for an internship under supervision of Santiago Zanella Béguelin.

References

[1] J. Groth and A. Sahai, "Efficient noninteractive proof systems for bilinear groups," *SIAM Journal on Computing*, vol. 41, no. 5, pp. 1193–1232, 2012.

[2] G. Barthe, J. M. Crespo, B. Grégoire, C. Kunz, Y. Lakhnech, B. Schmidt, and S. Zanella-Béguelin, "Fully automated analysis of padding-based encryption in the computational model," in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, ser. CCS '13. New York, NY, USA: ACM, 2013, pp. 1247–1260.

[3] S. D. Galbraith, K. G. Paterson, and N. P. Smart, "Pairings for cryptographers," *Discrete Applied Mathematics*, vol. 156, no. 16, pp. 3113 – 3121, 2008.

[4] J. Groth, "Simulation-sound NIZK proofs for a practical language and constant size group signatures," in *Advances in Cryptology ASIACRYPT 2006*, ser. Lecture Notes in Computer Science, X. Lai and K. Chen, Eds., vol. 4284. Springer Berlin Heidelberg, 2006, pp. 444–459.

[5] G. Fuchsbauer, "Automorphic signatures in bilinear groups and an application to round-optimal blind signatures," Cryptology ePrint Archive, Report 2009/320, 2009, http://eprint.iacr.org/.

[6] M. Abe, G. Fuchsbauer, J. Groth, K. Haralambiev, and M. Ohkubo, "Structure-preserving signatures and commitments to group elements," in *Advances in Cryptology - CRYPTO 2010*, ser. Lecture Notes in Computer Science, T. Rabin, Ed. Springer Berlin Heidelberg, 2010, vol. 6223, pp. 209–236.

[7] M. Abe, K. Haralambiev, and M. Ohkubo, "Signing on elements in bilinear groups for modular protocol design," Cryptology ePrint Archive, Report 2010/133, 2010, http://eprint.iacr.org/.

[8] M. Abe, J. Groth, K. Haralambiev, and M. Ohkubo, "Optimal structure-preserving signatures in asymmetric bilinear groups," in *Advances in Cryptology - CRYPTO 2011*, ser. Lecture Notes in Computer Science, P. Rogaway, Ed. Springer Berlin Heidelberg, 2011, vol. 6841, pp. 649–666.

[9] M. Abe, M. Chase, B. David, M. Kohlweiss, R. Nishimaki, and M. Ohkubo, "Constant-size structure-preserving signatures: Generic constructions and simple assumptions," in *Advances in Cryptology - ASIACRYPT 2012*, ser. Lecture Notes in Computer Science, X. Wang and K. Sako, Eds. Springer Berlin Heidelberg, 2012, vol. 7658, pp. 4–24.

[10] D. Hofheinz and T. Jager, "Tightly secure signatures and public-key encryption," in *Advances in Cryptology - CRYPTO 2012*, ser. Lecture Notes in Computer Science, R. Safavi-Naini and R. Canetti, Eds. Springer Berlin Heidelberg, 2012, vol. 7417, pp. 590–607.

[11] I. Visconti and R. Prisco, Eds., *A New Hash-and-Sign Approach and Structure-Preserving Signatures from DLIN*, ser. Lecture Notes in Computer Science, vol. 7485. Springer Berlin Heidelberg, 2012.

[12] J. Camenisch, M. Dubovitskaya, and K. Haralambiev, "Efficient structure-preserving signature scheme from standard assumptions," in *Security and Cryptography for Networks*, ser. Lecture Notes in Computer Science, I. Visconti and R. Prisco, Eds., vol. 7485. Springer Berlin Heidelberg, 2012, pp. 76–94.

[13] M. Abe, B. David, M. Kohlweiss, R. Nishimaki, and M. Ohkubo, "Tagged one-time signatures: Tight security and optimal tag size," in *Public-Key Cryptography PKC 2013*, ser. Lecture Notes in Computer Science, K. Kurosawa and G. Hanaoka, Eds., vol. 7778. Springer Berlin Heidelberg, 2013, pp. 312–331.

[14] M. Abe, J. Groth, M. Ohkubo, and M. Tibouchi, "Unified, minimal and selectively randomizable structure-preserving signatures," in *Theory of Cryptography*, ser. Lecture Notes in Computer Science, Y. Lindell, Ed., vol. 8349. Springer Berlin Heidelberg, 2014, pp. 688–712.

[15] S. Kremer, A. Mercier, and R. Treinen, "Reducing equational theories for the decision of static equivalence," in *Advances in Computer Science - ASIAN 2009. Information Security and Privacy*, ser. Lecture Notes in Computer Science, A. Datta, Ed. Springer Berlin Heidelberg, 2009, vol. 5913, pp. 94–108.

[16] F. Bobot, J.-C. Filliâtre, C. Marché, and A. Paskevich, "Why3: Shepherd your herd of provers," in *Boogie 2011: First International Workshop on Intermediate Verification Languages*, Wroclaw, Pologne, 2011, pp. 53–64.

[17] L. Moura and N. Bjørner, "Z3: An efficient SMT solver," in *Tools and Algorithms for the Construction and Analysis of Systems*, ser. Lecture Notes in Computer Science, C. Ramakrishnan and J. Rehof, Eds., vol. 4963. Springer Berlin Heidelberg, 2008, pp. 337–340.

[18] F. Bobot, S. Conchon, E. Contejean, M. Iguernelala, S. Lescuyer, and A. Mebsout. (2008) The Alt-Ergo automated theorem prover. [Online]. Available: http://alt-ergo.lri.fr

[19] C. Barrett and C. Tinelli, "CVC3," in *Computer Aided Verification*, ser. Lecture Notes in Computer Science, W. Damm and H. Hermanns, Eds., vol. 4590. Springer Berlin Heidelberg, 2007, pp. 298–302.

[20] B. Blanchet, "An efficient cryptographic protocol verifier based on Prolog rules," in *Computer Security Foundations Workshop (CSFW)*. IEEE, 2001, pp. 82–96.