Automatic Analysis of the Security of XOR-Based Key Management Schemes

Véronique Cortier[1,⋆], Gavin Keighren[2], and Graham Steel[2,⋆⋆]

[1] Loria UMR 7503 & CNRS & INRIA Lorraine projet Cassis, France
Veronique.Cortier@loria.fr
http://www.loria.fr/~cortier
[2] School of Informatics, University of Edinburgh, Scotland
Graham.Steel@ed.ac.uk
http://homepages.inf.ed.ac.uk/gsteel

Abstract. We describe a new algorithm for analysing security protocols that use XOR, such as key-management APIs. As a case study, we consider the IBM 4758 CCA API, which is widely used in the ATM (cash machine) network. Earlier versions of the CCA API were shown to have serious flaws, and the fixes introduced by IBM in version 2.41 had not previously been formally analysed. We first investigate IBM's proposals using a model checker for security protocol analysis, uncovering some important issues about their implementation. Having identified configurations we believed to be safe, we describe the formal verification of their security. We first define a new class of protocols, containing in particular all the versions of the CCA API. We then show that secrecy after an unbounded number of sessions is decidable for this class. Implementing the decision procedure requires some improvements, since the procedure is exponential. We describe a change of representation that leads to an implementation able to verify a configuration of the API in a few seconds. As a consequence, we obtain the first security proof of the fixed IBM 4758 CCA API with unbounded sessions.

1 Introduction

Security protocols are small programs that aim to secure communications over a public network like the Internet. The design of such protocols is notoriously difficult and error-prone. Formal methods have proved their usefulness in the rigorous analysis of security protocols. Methods developed for security protocol analysis can also be useful for analysing other security-critical designs: for example, the security APIs of hardware security modules (HSMs). HSMs are essentially cryptographic co-processors encased in tamper-proof enclosures, and are widely used in security critical systems such as electronic payment and automated teller machine (ATM) networks. Use of the HSM is governed by the API, which can be thought of as a set of two-party security protocols, each describing an exchange between the HSM and the user, which may be

⋆ This work has been partially supported by the ACI-SI project SATIN and the RNTL project POSE.

⋆⋆ Supported by EPSRC Grant number GR/S98139/01, 'Automated Analysis of Security Critical Systems'.

O. Grumberg and M. Huth (Eds.): TACAS 2007, LNCS 4424, pp. 538–552, 2007.

used in any order. The IBM 4758 CCA[1] is an important example of such an API. In 2001, Bond discovered flaws in the CCA key management scheme that allowed an intruder to obtain access to PINs [2, §5.1]. The attack requires the intruder to exploit the algebraic properties of the XOR operation, which is used extensively in the CCA. Bond proposed changes to the API, which have been shown to be secure, [8], but these changes would not have been backward-compatible. IBM made changes of their own in version 2.41 of the API, and provided several procedural recommendations to prevent the attack. Though previous formal work has been able to rediscover the flaws in the old version [12,15], the new version of the API had not been formally analysed before.

In this paper, we propose a thorough analysis of the security of the revised IBM protocol, combining a case study with the development of a new tool for analysing protocols with XOR. Our first main contribution is the analysis of the IBM recommendations using the CL-AtSe protocol analysis tool [13], during which we discovered a possible attack, that we reported to IBM (see §2.3). For all the other versions, CL-AtSe concludes that the IBM protocol is safe. However, CL-AtSe only checks security for a finite number of *sessions*, or runs of the protocol. Furthermore, because the complexity of the API is greater than a standard key-exchange protocol, the number of sessions checked is often very small, usually only three. This means that there is no guarantee of security if the protocol is executed more than three times. In addition, the IBM CCA API lies outside both the existing classes of protocols using XOR which have been previously shown to be decidable [5,14] for an unbounded number of sessions. Other decidable classes of protocols with XOR have been proposed [6,4] but they only model a finite number of sessions.

To address this problem, our second main contribution is the development of a new class of protocols, called WFX-class, that includes the IBM CCA API. We show this class to be decidable. Our proof is considerably simpler that the corresponding proofs for the two previously treated classes, but the resulting decision procedure still has an exponential complexity. For example, in our application, our decision procedure may require us to compute about 2^{26} terms. We describe a change of representation that leads to an implementation able to verify a configuration of the API in a few seconds. As a consequence, we obtain the first security proof of the fixed IBM 4758 CCA API with unbounded sessions. To the best of our knowledge, this implementation is the first tool for automatically verifying protocols with XOR, for an unbounded number of sessions, albeit for a particular class of protocols.

The paper is organised as follows: in §2, we analyse each of IBM's recommendations for patching the CCA key management protocol, using CL-AtSe (thus for a bounded number of sessions). In §3, we define our class of well-formed protocols, and prove the decidability of security of protocols in the class for an unbounded number of sessions. We explain how to implement our procedure in §4, and provide our results for the CCA key management scheme. Concluding remarks can be found in §5. A longer version of the paper, containing full proofs and more details of the CCA command modelling, has been issued as a technical report [7].

[1] CCA stands for 'Common Cryptographic Architecture', while 4758 is the model number of the HSM. See `http://www-3.ibm.com/security/cryptocards/pcicc.shtml`

2 Analysing the IBM Recommendations Using CL-AtSe

HSMs typically consist of a cryptoprocessor and a small amount of memory inside a tamper-proof enclosure. They are designed so that should an intruder open the casing or insert probes to try to read the memory, it will auto-erase in a matter of nanoseconds. In a typical ATM network application, HSMs are used, for example, to decrypt, encrypt and verify PINs. Many different keys may are used for these operations. IBM's Common Cryptographic Architecture (CCA) API [3] supports various key types, such as data keys, key encryption keys, import keys and export keys. Each type is represented by a public 'control vector' which is XOR-ed with the security module's master key (which is stored inside the HSM), before being used to encrypt the particular key. For example, a data key would be encrypted under $\texttt{KM}\oplus\texttt{DATA}$.[2] Keys encrypted in this manner are known as *working keys* and are stored outside of the security module. They can then only be used by sending them back into the HSM under the desired API command. Only particular types of keys will be accepted by the HSM for particular operations. For example, data keys can be used to encrypt arbitrary messages, but so-called 'PIN Derivation Keys' (PDKs, with control vector PIN) cannot, which is critical for security: a customer's PIN is just his account number encrypted under a PIN derivation key. In Bond's attack, the intruder uses API commands to change the type of a key, exploiting the algebraic properties of XOR. This allows a PIN derivation key to be converted into a data key, which can then be used to encrypt data. Hence the attack allows a criminal to generate a PIN for any account number. For more details of Bond's 'Chosen Key Difference' attack, see [2, §5.1].

2.1 CCA Key Management Commands

Following previous work [15,12], our experiments consider a number of key management commands from the CCA API. We ignore commands which do not generate key material and commands that are subsumed by more general ones. A full list of commands, including the ones not modelled and our justification for leaving them out, can be found in [7]. The modelled rules of the IBM 4758 CCA API are represented in Figure 1. For each command, the terms on the left of the arrow represent the user's input to the HSM, and the term on the right represents the HSM's output. For example, we have seen that data keys should be encrypted under $\texttt{KM}\oplus\texttt{DATA}$. Thus the **Encipher** rule corresponds to a data encryption command which allows data keys to be used to encrypt any given plaintext. **Decipher** allows data keys to be used for decryption. **Key Import** allows a key from another 4758 module, encrypted for transport under a 'key encrypting key' (KEK), to be made into a working key for this HSM. **Key Export** is used to encrypt a working key under a key encrypting key for transport to another HSM. Note the division of types of KEK: IMP for import and EXP for export. In order to transport encrypted keys to a new HSM, an importer KEK must first be established as a working key at the destination HSM. In order to do this without giving away the value of the KEK, which would be a considerable security risk, the KEK is decomposed into three parts, which XOR together to give the final KEK. The three **Key Part Import**

[2] $\oplus$ represents bitwise XOR.

x , $\{\!|xkey|\!\}_{\mathtt{KM}\oplus\mathtt{DATA}} \rightarrow \{\!|x|\!\}_{xkey}$ **Encipher**

$\{\!|x|\!\}_{xkey}$, $\{\!|xkey|\!\}_{\mathtt{KM}\oplus\mathtt{DATA}} \rightarrow x$ **Decipher**

$\{\!|xkey|\!\}_{xkek\oplus xtype}$, $xtype$, $\{\!|xkek|\!\}_{\mathtt{KM}\oplus\mathtt{IMP}} \rightarrow \{\!|xkey|\!\}_{\mathtt{KM}\oplus xtype}$ **Key Import**

$\{\!|xkey|\!\}_{\mathtt{KM}\oplus xtype}$, $xtype$, $\{\!|xkek|\!\}_{\mathtt{KM}\oplus\mathtt{EXP}} \rightarrow \{\!|xkey|\!\}_{xkek\oplus xtype}$ **Key Export**

$xkpNew$, $xtype \rightarrow \{\!|xkpNew|\!\}_{\mathtt{KM}\oplus xtype\oplus\mathtt{KPART}}$ **Key Part Import 1**

$xkpNew$, $xtype$, $\{\!|xkpOld|\!\}_{\mathtt{KM}\oplus xtype\oplus\mathtt{KPART}} \rightarrow \{\!|xkpNew\oplus xkpOld|\!\}_{\mathtt{KM}\oplus xtype\oplus\mathtt{KPART}}$
Key Part Import 2

$xkpNew$, $xtype$, $\{\!|xkpOld|\!\}_{\mathtt{KM}\oplus xtype\oplus\mathtt{KPART}} \rightarrow \{\!|xkpNew\oplus xkpOld|\!\}_{\mathtt{KM}\oplus xtype}$
Key Part Import 3

$\{\!|xkey|\!\}_{xkek1\oplus xtype}$, $xtype$, $\{\!|xkek1|\!\}_{\mathtt{KM}\oplus\mathtt{IMP}}$, $\{\!|xkek2|\!\}_{\mathtt{KM}\oplus\mathtt{EXP}} \rightarrow \{\!|xkey|\!\}_{xkek2\oplus xtype}$
Key Translate

Variables are prefixed by x. The term $\{\!|m|\!\}_k$ represents the message m encrypted with the key k (using symmetric encryption).

Fig. 1. Modelled rules of the IBM 4758 CCA API

commands can then be used one after the other, by three different security officers, each in possession of one key part, to create the working import key. It is this process that is subverted in Bond's attack to change the type of a key. **Key Translate** is used to translate a key from encryption under one KEK (of import type) to encryption under another (of export type). For a full description of all these rules, see [10].

2.2 Modelling the API

We chose to use CL-AtSe [13] to check the API since unlike most protocol analysis tools, it has built-in support for the XOR operator. CL-AtSe is a 'Dolev-Yao'[3] style protocol analyser, part of the AVISPA tool set [16]. It accepts models written in a special-purpose protocol specification language called HLPSL [17], and implements a variant of the Baader-Schulz unification algorithm [1], optimised for XOR. The HLPSL is initially converted into a transition relation, which CL-AtSe uses to generate a set of constraints describing the protocol. Each protocol step is modelled by constraints on the intruder's knowledge, with the execution of such steps simulated by adding new constraints to the system and by reducing or eliminating existing constraints. Security properties are checked against the system state at each step. See [17, §3.2.1] for more details of the operation of CL-AtSe.

Each of the commands were modelled as a separate 'role' containing exactly one transition. The intruder's initial knowledge includes an unknown working key of each

[3] This refers to the nature of the intruder being modelled, who may decompose and re-assemble message parts, but not perform any cryptanalytic attacks.

type, to reflect that fact that even if he does not already have such keys, he can always 'conjure' one by repeatedly trying random values against a command, [2, §3.4]. In addition, he is given all the initial knowledge assumed by Bond in his attacks, [2], which includes a key part K3, a partially completed importer key $\{\!|\mathtt{KEK}\oplus\mathtt{K3}|\!\}_{\mathtt{KM}\oplus\mathtt{KP}\oplus\mathtt{IMP}}$, a PIN derivation key PDK encrypted under transport key KEK, and a customer's account number PAN. For standard security protocols, we would be interested in model checking properties such as the secrecy of a newly agreed session key, i.e. that a term representing the session key is unknown to the intruder. In the case of security APIs, we are interested in the secrecy of the cleartext value of the sensitive keys managed by the HSM. Additionally, we assume that the intruder knows a customer's account number, since these are not kept secret in the system. We can now also check the secrecy of the customer's PIN, i.e. the account number encrypted under the PIN derivation key, that is a message of the form $\{\!|\mathtt{PAN}|\!\}_{\mathtt{PDK}}$. This accounts for attacks where the intruder is able to encrypt arbitrary data under the PDK, without learning the key's cleartext value, as is the case in several of Bond's attacks.

Full details of the CL-AtSe modelling can be found in [10]. Having established that CL-AtSe can very quickly re-discover Bond's attack on the original API, we proceeded to investigate IBM's recommendations for preventing it.

2.3 Analysing IBM's Recommendations

In response to Bond's attacks [2, §5.1], IBM released a set of three recommendations designed to prevent it [9], covering command usage, the access control system, and general procedural safeguards. However, it was unclear which of the recommendations are necessary, or sufficient, to prevent the attack. We investigated all the recommendations using our CL-AtSe model.

Recommendation 1 – Use Public Key Techniques. Instead of transferring the initial key encryption key (KEK) using key parts in clear, IBM recommend that it is transferred encrypted under the destination HSM's public key. This ensures that the KEK is never present in clear, and thus cannot be modified. Using this method, the KEK is wrapped in a key block which is subsequently encrypted and provided as input to the **PKA Symmetric Key Import** command, defined as follows:

$$\{\!|xkey.xtype|\!\}_{\mathtt{PK}} \rightarrow \{\!|xkey|\!\}_{\mathtt{KM}\oplus xtype} \qquad \textbf{PKA Symmetric Key Import}$$

However, the format and encryption procedure for the key block is given in the manual, and it is therefore possible for a block containing an arbitrary key to be created, thus allowing a known key to be introduced into the security module. CL-AtSe quickly discovered that an attacker could introduce a known exporter key k,[4] and obtain the transported PIN derivation key encrypted under this key (see Figure 2). We reported this vulnerability to IBM. They conceded that the attack was possible, and intend to change the documentation to reflect this. They argue the attack would have to be carried out by an insider, and that the vulnerability is intrinsic to public key schemes. We

[4] Our experiments found that, even if the **PKA Symmetric Key Import** command does not accept export-type KEKs, it is still possible to obtain such a key (see [10] for details).

$\{\!|kek.\texttt{IMP}|\!\}_{\texttt{PK}} \rightarrow \{\!|kek|\!\}_{\texttt{KM}\oplus\texttt{IMP}}$ **PKA Symmetric Key Import**

$\{\!|k.\texttt{EXP}|\!\}_{\texttt{PK}} \rightarrow \{\!|k|\!\}_{\texttt{KM}\oplus\texttt{EXP}}$ **PKA Symmetric Key Import**

$\{\!|pdk|\!\}_{kek\oplus\texttt{PIN}}, \texttt{PIN}, \{\!|kek|\!\}_{\texttt{KM}\oplus\texttt{IMP}} \rightarrow \{\!|pdk|\!\}_{\texttt{KM}\oplus\texttt{PIN}}$ **Key Import**

$\{\!|pdk|\!\}_{\texttt{KM}\oplus\texttt{PIN}}, \texttt{PIN}, \{\!|k|\!\}_{\texttt{KM}\oplus\texttt{EXP}} \rightarrow \{\!|pdk|\!\}_{k\oplus\texttt{PIN}}$ **Key Export**

Fig. 2. A known-exporter attack. The attacker first imports the import-type KEK as intended, then imports an export-type key k which he knows. Then, he imports the PDK as intended, but then can export it under $k\oplus\texttt{PIN}$, and since PIN is a public value, he can decrypt this packet and obtain the PDK.

suggest that access control should be used to restrict any single insider from having access to both the **PKA Symmetric Key Import** and **Key Import** commands. We created two models, each one allowing access to only one of these functions, and checked them with CL-AtSe, which discovered no further attacks, up to the bounds shown in the table below:

Available Command	Bound*	Analysed States	Reachable States	Run-Time (s)
Key Import	$10^{\#}$	76	10	0.08
PKA Symmetric Key Import	3	8751	1749	514.27

$^{\#}$ This bound could be set much higher, but informal analysis showed that the intruder was never able to obtain any useful new terms.

* Bound on the number of sessions.

Recommendation 2 – Use the Access Control System. Users of IBM's 4758 HSM are assigned to roles that determine which commands they are allowed to execute. The goal is to prevent one single individual from having access to all the commands required to mount Bond's attack. This is enforced using access controls. IBM provide an example of the KEK transfer process involving five roles (A – E) such that no single role is able to mount the attack (see Figure 3).

In the original attack, the intruder played the roles C and E together. Note that roles A and D do not have any access privileges at the destination security module. IBM state in their recommendation that roles A and B could actually be played by the same individual. This does not hold since that person has access to all the key parts, and thus the completed KEK, so she could decrypt the key in transit, and obtain its clear value.

In our experimental model, the intruder was actually given a greater range of API commands than as suggested by IBM, with still the restriction that at least one of the three requirements for the attack were missing. That is, none of them gave the intruder access to a **Key Part Import** command, the **Key Import** command, and the key being transferred. The reason for this was that we were trying to discover the minimum

Person	Responsibilities	Commands
A	Generates and distributes the clear key parts, as well as the key verification pattern (KVP) for the complete KEK.	N/A
B	Enters the first key part into the destination security module.	Key Part Import 1
C	Enters the second key part, and verifies that the completed KEK is correct by checking the KVP.	Key Part Import 3 Key Test
D	Distributes the PIN derivation key (PDK) encrypted under KEK.	N/A
E	Verifies that the KEK is correct, then imports the PDK.	Key Test Key Import

Fig. 3. Roles described by IBM in their 2nd recommendation

restrictions that are sufficient to prevent the attack. CL-AtSe reported no attacks up to the bounds shown below:

Person	Bound	Analysed States	Reachable States	Run-Time (s)
B	6	34	6	333.02
C	3	413	68	58.22
E	10*	54	10	0.03

* This bound could be set much higher, but informal analysis showed that the intruder was never able to obtain any useful new terms.

Recommendation 3 – Use Procedural Controls. IBM's third recommendation is to ensure that no single individual involved in the key transfer process has the opportunity to modify the KEK used. If the KEK is not modified, then the type of the key being transferred cannot be altered when it is imported. With respect to the API commands, this translates to restricting the **Key Import** command to only accept the unmodified KEK. CL-AtSe found no attack on this version of the API:

Bound	Analysed States	Reachable States	Run-Time (s)
3	13133	2625	2827.35

The large number of reachable states reflects the fact that the intruder is still able to generate a large number of modified KEKs, even though they cannot be used to import the PDK. IBM now seem to intend that Recommendation 3 is always followed, in addition to any of the other recommendations, in order to ensure a high level of security. The points outlined by the recommendation have since been expanded and included in the current version of the CCA Manual [3, Appendix H] as general principles for secure operation.

All the model files used in our experiments are available from `http://homepages.inf.ed.ac.uk/gsteel/CCA-experiments/`. The CL-AtSe tool may be downloaded from `http://www.avispa-project.org/`.

3 Theoretical Results for XOR-Based Key-Management APIs

Having investigated IBM's recommendations with a model checker, and adjusting them where necessary to produce what seemed to be secure configurations, we proceed towards verifying them secure. As we have seen, both protocols and intruder behaviours can be modelled symbolically using rules over terms with variables. We observe that the IBM 4758 CCA API can actually be modelled using what we call *well-formed* rules. We then show that reachability of a term is decidable for any set of well-formed rules.

3.1 Definitions

Cryptographic primitives are represented by functional symbols. More specifically, we consider the *signature* Σ containing an infinite number of constants including some special constant 0 and two non constant symbols $\{\!|\_|\!\}\_$ and $\oplus$ of arity 2. We also assume an infinite set of variables $\mathcal{V}$. The set of *terms or messages* is defined inductively by

$T ::=$		terms
	x	variable x
$\mid$	$f(T_1, \ldots, T_k)$	application of symbol $f \in \Sigma$ of arity $k \geq 1$
$\mid$	c	constant $c \in \Sigma$

A term is *ground* if it has no variable.

As in §2, the term $\{\!|\, m \,|\!\}_k$ is intended to represent the message m encrypted with the key k (using symmetric encryption). The term $m_1 \oplus m_2$ represents the message m_1 XORed with the message m_2. The constants may represent agent identities, nonces or keys for example. Substitutions are written $\sigma = \{x_1 = t_1, \ldots, x_n = t_n\}$ with $dom(\sigma) = \{x_1, \ldots, x_n\}$. σ is *ground* iff all of the t_i are ground. The application of a substitution σ to a term t is written $\sigma(t) = t\sigma$. The *size* of a term t, denoted by $|t|$, is defined as usual by the total number of symbols used in t. More formally, $|a| = 1$ if a is a constant or a variable and $|f(t_1, \ldots, t_n)| = 1 + \sum_{i=1}^{n} |t|$ if f is of arity $k \geq 1$. The size of a set of terms S is the sum of the size of the terms in S.

We equip the signature with an equational theory E that models the algebraic properties of the XOR operator:

$$x \oplus (y \oplus z) = (x \oplus y) \oplus z \qquad x \oplus y = y \oplus x$$
$$x \oplus x = 0 \qquad x \oplus 0 = x$$

It defines an equivalence relation that is closed under substitutions of terms for variables and under application of contexts. In particular, we say that two terms t_1 and t_2 are equal, denoted by $t_1 = t_2$ if they are equal modulo the equational theory E. If two terms are equal using only the equations of the first line, we say that they are equal modulo Associativity and Commutativity (AC).

Intruder capabilities and the protocol behaviour are described using *rules* of the form $t_1, \ldots, t_n \rightarrow t_{n+1}$ where the t_i are terms.

Example 1. The intruder capabilities are represented by the following set of three rules:

$$\begin{array}{rl} x, y \rightarrow \{\!|x|\!\}_y & \text{encryption} \\ \{\!|x|\!\}_y, y \rightarrow x & \text{decryption} \\ x, y \rightarrow x \oplus y & \text{xoring} \end{array}$$

The set of deducible terms is the reflexive and transitive closure of the rewrite rules.

Definition 1. *Let $\mathcal{R}$ be a set of rules. Let S be a set of ground terms. The term u is* one-step deducible *from S if there exists a rule $t_1, \ldots, t_n \rightarrow t \in \mathcal{R}$ and a ground substitution θ such that $t_i\theta \in S$ and $u = t\theta$.*

A term u is deducible *from S, denoted by $S \vdash_{\mathcal{R}} u$, if $u \in S$ or there exist ground terms $u_1, \ldots, u_n$ such that $u_n = u$ and u_i is one-step deducible from $S \cup \{u_1, \ldots, u_{i-1}\}$ for every $1 \leq i \leq n$. The sequence $u_1, \ldots, u_n$ is a* proof *that $S \vdash_{\mathcal{R}} u$.*

We write $\vdash$ instead of $\vdash_{\mathcal{R}}$ when $\mathcal{R}$ is clear from the context.

Example 2. Let $\mathcal{R}$ be the set of rules described in Example 1. Let $S = \{\{\!|\, n \,|\!\}_a, a \oplus b, b\}$. Then n is deducible from S and $\{\!|\, n \,|\!\}_a, a \oplus b, b, a$ is a proof of $S \vdash n$. Indeed a is one-step deducible from $\{a \oplus b, b\}$ using the rule $x, y \rightarrow x \oplus y$ and the fact that $(a \oplus b) \oplus b = a$ and n is one-step deducible from $\{\{\!|n|\!\}_a, a\}$ using the rule $\{\!|x|\!\}_y, y \rightarrow x$.

3.2 Well-Formed Protocols

Rather than restricting the use of variables in protocol rules, we take advantage of the form of API-like protocols, noticing that they only perform simple operations.

Definition 2. *A term t is an* XOR *term if $t = \bigoplus_{i=1}^{n} u_i$, $n \geq 1$ where each u_i is a variable or a constant.*

A term t is an encryption *term if $t = \{\!|u|\!\}_v$ where u and v are XOR terms.*

A term t is a well-formed term *if it is either an encryption term or an XOR term. In particular, a well formed term contains no nested encryption.*

A rule $t_1, \ldots, t_n \rightarrow t_{n+1}$ is well formed *if*

- *each t_i is a well-formed term.*
- *$Var(t_{n+1}) \subseteq \bigcup_{i=1}^{n} Var(t_i)$ (no variable is introduced in the right-hand-side of a rule).*

A proof is well-formed *if it only uses well-formed terms.*

Definition 3. *The* WFX-class protocol *consists of a pair $(\mathcal{R}, S)$, where $\mathcal{R}$ is a finite set of well-formed rules, and S is a finite set of ground, well-formed terms.*

Intuitively, the rules in $\mathcal{R}$ represent the commands of the API and the intruder capabilities, and the ground terms S the initial knowledge of the intruder. We call our class WFX since these are well-formed protocols using the XOR operator. In particular, the rules representing the intruder capabilities (defined in Example 1) and the rules representing the 4758 CCA API protocol (introduced in §2 are all well-formed.

The remaining of the section is devoted to the decidability of deducibility of a term, which can be used to encode secrecy preservation of a protocol. To the best of our knowledge, there exist only two decidable classes [5,14] for secrecy preservation for protocols with XOR, for an unbounded number of sessions. In both cases, the main difference with our class is that we make restrictions on the combination of functional symbols rather than on the occurrences of variables. As a consequence, our class is incomparable to the two existing ones. A more detailed discussion may be found in [7].

3.3 Proof of Decidability

The key idea of our decidability result is to show that only well-formed terms need to be considered when checking for the deducibility of a (well-formed) term. In particular, there is no need to consider nested encryption. This allows us to consider only a finite number of terms: we have a finite number of atoms in the initial set of rules which can only be combined by encryption and XORing, and XORing identical atoms results in cancellation. At the end of the proof, we comment on the complexity of the resulting decision procedure.

We first prove that whenever an encryption occurs in a deducible term, the encryption is itself deducible.

Proposition 1. *Let $\mathcal{R}$ be a set of well-formed rules. Let S be a set of ground well-formed terms (intuitively the initial knowledge). Let u be a term such that $S \vdash u$ and let $\{\!|u_1|\!\}_{u_2}$ be a subterm of u. Then $S \vdash \{\!|u_1|\!\}_{u_2}$.*

The proof is by induction on the number of steps needed to obtain u. The full proof is in [7].

Our main result states that only well-formed terms need to be considered when checking for deducibility.

Proposition 2. *Let $\mathcal{R}$ be a set of well-formed rules and S be a set of ground well-formed terms such that*

- *$\mathcal{R}$ contains the rule $x, y \rightarrow x \oplus y$;*
- *S contains 0 (the null element for XOR should always be known to an intruder).*

Let u be a ground well formed term deducible from S. Then there exists a well-formed proof of $S \vdash u$.

We briefly sketch the proof of this key proposition (the full proof appears in [7]). Taking advantage of the form of the rules, the main idea is to show that, considering a proof of a well-formed term u and removing all inside encrypted terms, we obtain a (well-formed) proof of u. We define a function $t \mapsto \overline{t}$ that removes inside encryption. For example, we have $\overline{\{\!|a \oplus \{\!|a|\!\}_b|\!\}_c \oplus \{\!|c|\!\}_b} = \{\!|a|\!\}_c \oplus \{\!|c|\!\}_b$. Roughly, we show by induction on the length of the proof that whenever $u_1, \ldots, u_n$ is a proof then $\overline{u_1}, \ldots, \overline{u_n}$ is a proof. Assume $u_1, \ldots, u_n, u_{n+1}$ is a proof and $t_1, \ldots, t_k \rightarrow t$ is the last rule been applied. There is a substitution θ such that $t\theta = u_{n+1}$ and $t_i\theta = u_{j_i}$. Since t is a well-formed term, any inside encryption e of u_{n+1} must appear under a variable x in t thus e also appears in some u_{j_i}. Intuitively, there is a case analysis depending on whether x also appears

under an encryption in t_i. If x does not appear under an encryption, that is $t = x \oplus t'$, we use the fact that (Proposition 1) e is deducible thus $u_{j_i} \oplus e$ is also deducible and we could have chosen $x\theta' = x\theta \oplus e$, removing the encryption from u_{n+1}.

Using Proposition 2, we can now easily conclude the decidability of deducibility.

Theorem 1. *The following problem*

- *Given a finite set of well-formed rules $\mathcal{R}$ containing the rule $x, y \rightarrow x \oplus y$, a finite set S of ground well-formed terms containing 0 and a ground well-formed term u,*
- *Does $S \vdash_{\mathcal{R}} u$?*

is decidable in exponential time in the size of $\mathcal{R}$, S and u.

Let $a_1, \ldots, a_n$ be the constants that occur in $\mathcal{R}$, S or u. Let k be the maximal number of terms in the left-hand side of a rule in $\mathcal{R}$. For any $t_1, \ldots, t_l \rightarrow t \in \mathcal{R}$, we have $l \leq k$. We show that $S \vdash_{\mathcal{R}} u$ can be decided in $\mathcal{O}(2^{(k+1)(2n+1)})$.

The decision procedure is as follows: we saturate S by adding any well-formed deducible terms. We obtain a set S^*. By Proposition 2, $S \vdash_{\mathcal{R}} u$ if and only if $u \in S^*$. In S^* there are at most

- 2^n XOR terms
- and $2^n \times 2^n = 2^{2n}$ encryption terms

thus $|S^*| \leq 2^{2n+1}$. Note that we consider here terms modulo AC which means that we only consider one concrete representation for each class of terms equal modulo AC. This can be done for example by fixing an arbitrary order on the constants and using it to normalise terms.

Now, at each iteration, For each rule $t_1, \ldots, t_l \rightarrow t \in \mathcal{R}$ we consider any tuple of terms $(u_1, \ldots, u_l)$ with u_i in the set that is being saturated and compute the set $\mathcal{M}$ of most general unifiers of $(u_1, \ldots, u_l) = (t_1, \ldots, t_l)$ (which can be done in polynomial time for well-formed terms, see [7]). Then we add any well-formed instance of $t\sigma$ for any $\sigma \in \mathcal{M}$. We consider at most $|S^*|^k \leq 2^{k(2n+1)}$ tuples at each iteration. All together, we need at most $\mathcal{O}(2^{(k+1)(2n+1)})$ operations to compute S^*.

4 Implementation and Results

Our efforts to implement the decision procedure using existing tools such as theorem provers (Vampire and E) and model finders (Paradox and Darwin) were unsuccessful. The combinatorial complexity caused by the XOR operation prevents any of the tools from finding a saturation. Since our models have a finite Herbrand universe, and hence are effectively propositional, we considered a manual encoding as a Boolean satisfiability problem, for use with a SAT solver. Unfortunately, for n atoms (our models typically have $n = 13$), we will need 2^n (possible XOR terms) + $2^n \times 2^n$ (possible encryption terms) propositional variables to represent the intruder's knowledge. Additionally, writing out ground versions of the 8 well formed rules in the API will result is an enormous problem, far too large for any SAT solver. In the end, we solved the problem by making a change of representation, and writing an ad-hoc decision procedure for that representation.

4.1 Representation of XOR Terms

The representation consists of encoding an XOR term as a binary string, accomplished by assigning an (arbitrary) order to the finite set of atoms (or *base terms*). For example, if we have the ordered set of base terms KM, KP, KEK, IMP, EXP, DATA, PIN, we would represent KEK⊕PIN⊕DATA as

	KM	KP	KEK	IMP	EXP	DATA	PIN
KEK⊕PIN⊕DATA →	0	0	1	0	0	1	1

↓

19

Hence KEK⊕PIN⊕DATA is represented by the decimal integer 19. Notice the order of the atoms in the term does not matter - we still get the same integer - so our representation effectively normalises the term with respect to the properties of XOR. Notice further that if we have two terms $x1$ and $x2$, that are represented by integers l and m, then the integer representing $x1 \oplus x2$ is just $l \oplus m$. So, we represent XOR using XOR, which is an attractive feature of the representation. For example, we can write the intruder rule

$$x1, x2 \rightarrow x1 \oplus x2$$

as

$$l, m \rightarrow l \oplus m$$

For encryption terms, which consist of one XOR term encrypted by another, we simply shift the bits of the integer representing the message term n places to the left (where n is the number of base terms), and add the integer representing the key. We obtain a unique number in the range $0 \ldots 2^{2n}$ for each encryption term. For example, the term $\{\!|\text{KEK}\oplus\text{PIN}\oplus\text{DATA}|\!\}_{\text{KM}\oplus\text{DATA}}$ is represented by

KM	KP	KEK	IMP	EXP	DATA	PIN		KM	KP	KEK	IMP	EXP	DATA	PIN
0	0	1	0	0	1	1		1	0	0	0	0	1	0

↓

2498

4.2 The Implemented Procedure

Our decision procedure starts by allocating enough space in memory for $2^{2n} + 2^n$ integers, and setting all these memory locations to 0. Then, all locations corresponding to the intruder's initial knowledge $\mathcal{S}$ are set to 1, indicating that the intruder can obtain these terms. For each rule r_i in $\mathcal{R}$, with k terms on the left hand side, encode the operation as a partial function $f_i : \mathbb{N}^k \rightarrow \mathbb{N}$. As a simple example, for the 'Encipher' rule:

x , $\{\!|xkey|\!\}_{\text{KM}\oplus\text{DATA}} \rightarrow \{\!|x|\!\}_{xkey}$ **Encipher**

Assuming KM⊕DATA is represented by the integer value p, we write

$$f : x, [xkey|p], \rightarrow [x|xkey]$$

where the braces [|] denote composition of the two n-long bitstrings into a single $2n$-long bitstring. A more complicated example is the **Key Import** command:

$$\{|xkey|\}_{xkek \oplus xtype}, xtype, \{|xkek|\}_{\mathtt{KM} \oplus \mathtt{IMP}} \rightarrow \{|xkey|\}_{\mathtt{KM} \oplus xtype} \qquad \textbf{Key Import}$$

Assuming KM⊕IMP is represented by the integer q, and KM is represented by r, we write

$$f : [xkey|x], xtype, [xkek|q] \rightarrow [xkey|q \oplus xtype] \quad \text{IF} \quad x = xkek \oplus xtype$$

It will always be possible to write WFX class API rules as integer functions in this way provided the rules are executable, that is provided the HSM itself can work out the values of the bitstrings it needs to carry out the XORing or encryption/decryption required by the command. This leads directly to the integer formula required.

To obtain the fixpoint of the intruder's knowledge, we apply each rule exhaustively, looking for combinations of k suitable integers that the intruder already knows, and setting to 1 any location that we can now reach using these rules. We do this for all the rules in an iterative manner until no more rules apply. We check to see if any of the secret terms are now set to 1. If so, we have found an attack. If not, we have verified the API secure. Note that in the case where we find an attack, we cannot immediately return the trace of steps required to obtain the secret term, as CL-AtSe can. It would be possible to extend our procedure to keep track of the operations required to obtain each term, for example by outputting a list of terms obtained and post-processing the list to obtain the trace for the attack.

At each iteration of our decision procedure, it is possible to obtain terms we have already deduced, by repeating the original command application which returned the term in the first place. To avoid rediscovering existing terms, we mark the freshly obtained terms at each iteration, and require that a rule is applied only if it makes use of at least one fresh term. One final feature of our procedure is that is allows us to treat the value of the DATA control vector as zero, since this is its actual value, a fact which is exploited in an attack on the unrevised API presented by IBM themselves [9].

The full source code for our decision procedure, together with documentation and the files used for the experiments below, can be downloaded from `http://homepages.inf.ed.ac.uk/gsteel/CCA-experiments/`.

4.3 Results

Our WFX class does not account for public key encryption, nor the concatenation of key and type required by the **PKA Symmetric Key Import** command. So we modelled recommendation 1 by effectively pre-processing this command. We observe that the encrypted key blocks which it imports can either be legitimate (i.e. the intended KEK), or generated by the intruder from known unencrypted terms. Since the only operation the intruder can perform on the legitimate block is to execute **PKA Symmetric Key Import** on it, we provide him with the result of this, $\{|\, xKey \,|\}_{\mathtt{KM} \oplus xType}$, in his initial knowledge. This means that the **PKA Symmetric Key Import** command can be modelled such that it will only consider ways in which the intruder could use it to import self-generated encrypted blocks. Such blocks consist of a known unencrypted term and

Table 1. Results using our decision procedure to verify the recommendations

Model	Base Terms	Iterations	Terms Derived	Run-Time
Recommendation1_KeyImp	13	3	17015	0.23
Recommendation1_SymKeyImp	11	3	13045	3.04
Recommendation2_PersonB	14	2	4473	8.09
Recommendation2_PersonC	14	3	4413	12.10
Recommendation2_PersonE	13	2	1089	2.02
Recommendation3	14	3	83317	1.16

a key type control vector, so the command just becomes a way to turn a known unencrypted term into a working key of any type, i.e. the rule:

$xkey, xtype \rightarrow \{\!|xkey|\!\}_{\mathtt{KM} \oplus xtype}$ **Pre-Processed PKA Symmetric Key Import**

Apart from this change, all our models have the same initial knowledge and security goals as the CL-AtSe models. Table 1 summarises our results. We conclude that after our modifications described in §2.3 have been made, any one of the three recommendations is sufficient to secure the scheme against Dolev-Yao intruder attacks.

5 Conclusion

We have obtained a new decidable class of security protocols with XOR, for an unbounded number of sessions. The decision procedure has been implemented, yielding the first tool for automatically analysing a protocol with XOR and an unbounded number of sessions. As a case study, we have formally analysed the revised IBM 4758 CCA API protocol. We first discovered possible attacks using CL-AtSe, and refined IBM's recommendations to produce safe configurations. Our decision procedure then verified these configurations.

Related work includes that of Courant, who verified Bond's own suggestions for fixing the API in the interactive theorem prover Coq, [8]. His proof used normalisation functions to deal with XOR, and most of the proof effort was in showing these functions to be sound. Other work has looked at rediscovering Bond's attacks on the old API, [12,15], the latter work using (without proof) a heuristic that splits intruder knowledge into an encrypted and unencrypted part. We believe that our theoretical results show that their heuristic preserves attack-completeness.

In future we intend to try to extend our theoretical results to deal with asymmetric cryptography and pairing, so that we can analyse public-key management schemes, and to establish a formal theory that deals with so-called 'key-conjuring', [2, §3.4].

References

1. F. Baader and K. Schulz. Unification in the Union of Disjoint Equational Theories: Combining Decision Procedures. In D. Kapur, editor, *CADE-11: Eleventh International Conference on Automated Deduction*, volume 607, pages 50–65, June 1992.
2. M. Bond. Attacks on cryptoprocessor transaction sets. In Ç. K. KoÇ, D. Naccache, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2001*, volume 2162 of *Lecture Notes in Computer Science*, pages 220–234. Springer, 2001.

3. *CCA Basic Services Reference and Guide*, October 2006. Available online at `http://www-03.ibm.com/security/cryptocards/pdfs/bs327.pdf`.
4. Y. Chevalier, R. Küsters, M. Rusinowitch, and M. Turuani. An NP decision procedure for protocol insecurity with XOR. In *Proc. of 18th Annual IEEE Symposium on Logic in Computer Science (LICS '03)*, pages 261–270, 2003.
5. H. Comon-Lundh and V. Cortier. New decidability results for fragments of first-order logic and application to cryptographic protocols. In *Proceedings of the 14th International Conference on Rewriting Techniques and Applications (RTA'2003)*, volume 2706 of *LNCS*, pages 148–164, Valencia, Spain, June 2003. Springer-Verlag.
6. H. Comon-Lundh and V. Shmatikov. Intruder deductions, constraint solving and insecurity decision in presence of exclusive or. In *Proc. of 18th Annual IEEE Symposium on Logic in Computer Science (LICS '03)*, pages 271–280, 2003.
7. V. Cortier, G. Keighren, and G. Steel. Automatic analysis of the security of XOR-based key management schemes. Inf. Research Report EDI-INF-RR-0863, U. of Edinburgh, 2006.
8. J. Courant and J.-F. Monin. Defending the bank with a proof assistant. In *Proceedings of Workshop on Issues in the Theory of Security (WITS '06)*, Vienna, March 2006.
9. IBM Comment on "A Chosen Key Difference Attack on Control Vectors", January 2001. Available from `http://www.cl.cam.ac.uk/~mkb23/research.html`.
10. G. Keighren. Model checking IBM's common cryptographic architecture API. Informatics Research Report EDI-INF-RR-0862, University of Edinburgh, 2006.
11. R. Nieuwenhuis, editor. *Automated Deduction - CADE-20, 20th International Conference on Automated Deduction, Tallinn, Estonia, July 22-27, 2005, Proceedings*, volume 3632 of *Lecture Notes in Computer Science*. Springer, 2005.
12. G. Steel. Deduction with XOR constraints in security API modelling. In Nieuwenhuis [11], pages 322–336.
13. M. Turuani. The CL-Atse Protocol Analyser. In *Proceedings of the 17th International Conference on Rewriting Techniques and Applications (RTA'06)*, volume 4098 of *Lecture Notes in Computer Science*, pages 277–286, Seattle, WA, USA, 2006.
14. K. N. Verma, H. Seidl, and T. Schwentick. On the complexity of equational Horn clauses. In Nieuwenhuis [11], pages 337–352.
15. P. Youn, B. Adida, M. Bond, J. Clulow, J. Herzog, A. Lin, R. Rivest, and R. Anderson. Robbing the bank with a theorem prover. Technical Report UCAM-CL-TR-644, University of Cambridge, August 2005.
16. AVISPA Tool Set. Available from `http://www.avispa-project.org/`.
17. *AVISPA User Manual, version 1.1*, June 2006. Available online at `http://www.avispa-project.org/package/user-manual.pdf`.