STEVENS

Institute of Technology

Post-Quantum Cryptography Using Complexity

Doctoral Dissertation

Michael de Mare
Department of Computer Science
Stevens Institute of Technology
Castle Point on Hudson
Hoboken, NJ 07030

mdemare@cs.stevens.edu

Committee
Rebecca N. Wright
Stephen L. Bloom
Susanne Wetzel
Robert Gilman
Josh Benaloh

Final Version (December 1, 2009). Please do not distribute.

Secure Set Membership Using 3SAT

by

Michael James de Mare

A DISSERTATION

Submitted to the Faculty of the Stevens Institute of Technology in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Michael de Mare, Candidate	Date
ADVISORY COMMITTEE	
Rebecca N. Wright, co-chair	Date
Stephen L. Bloom, co-chair	Date
Susanne Wetzel	Date
Robert Gilman	Date
Josh Benaloh	Date

STEVENS INSTITUTE OF TECHNOLOGY
Castle Point on Hudson
Hoboken, NJ 07030
2009

©2009 Michael J. de Mare. All rights reserved.

SECURE SET MEMBERSHIP USING 3SAT

ABSTRACT

In this thesis, we develop a cryptographic primitive: a solution to the secure set membership problem. The secure set membership problem is the problem of creating a set representation so that it is possible to verify that a string is a member of the set without being able to learn members of the set from the representation. The secure set membership primitive includes a distributed protocol for set establishment and a proof-of-possession protocol to show set membership without revealing the member of the set.

We use a subset of the instances of the 3SAT witness-finding problem in developing a new complexity assumption for the secure set membership primitive. It is risky to base cryptosystems on *NP*-complete problems, as what is hard in the worst case may be easy in most cases. To mitigate this risk, we limit the instances of 3SAT that we generate to try to select only hard instances; our complexity assumption is different than mere *NP*-completeness.

We assume that 3SAT instances with certain parameters are difficult to find witnesses for. We back up this assumption with experiments using the MiniSAT SAT solver as well as our own software. We use these experiments to determine which parameters are most likely to yield a secure system.

We suggest applications of secure set membership including anonymous digital credentials, accounts with multiple users, access control lists, and document timestamping. We include both centralized and distributed protocols for establishing the set, and both pseudonomynous and anonymous protocols for verifying that a user possesses a witness that is a member of the set.

Author: Michael de Mare

Adviser: Rebecca N. Wright

Date: September 23, 2009

Department: Computer Science

Degree: Doctor of Philosophy

Acknowledgments

I would first and foremost like to thank my adviser, Rebecca Wright, without whom there would be no thesis. Her guidance and comments are what made this possible. I also appreciate her close readings of the drafts and comments that made the thesis what it is. I would also like to thank the rest of my committee, Stephen Bloom, Susanne Wetzel, Robert Gilman, and Josh Benaloh for their time and assistance. I would like to particularly thank Rebecca Wright and Robert Gilman for funding the research and development of this thesis. I would additionally like to thank Josh Benaloh for serving as my mentor, encouraging me to go to graduate school, providing guidance on where to go, and making it possible for me to attend Stevens.

I would like to thank my colleagues in my adviser's Tuesday afternoon meetings, Rebecca Wright, Vijay Ramachandran, Sotiris Ioannidis, Mike Engling, Onur Kardes, and Geetha Jagannathan for both their presentations and for listening to mine as well as their helpful comments. I would like to thank the members of the Laboratory for Secure Systems and the Algebraic Cryptography Center for listening to my work and offering their suggestions and comments.

I would like to thank the people with whom I spoke with at conferences and workshops, too numerous to mention, who asked insightful questions and provided helpful information. I would like to thank Whit Diffie for his comments on

symmetric-key ciphers and Lincoln Decoursey for help running experiments.

I would like to thank my parents for their (sometimes financial) support. I would particularly like to thank my mother for proofreading drafts of my thesis. This wouldn't have been possible without them.

Contents

List of Figures

Chapter 1

Introduction

In order to cope with new technologies such as quantum computing and the possibility of developing new algorithms, new cryptosystems should be developed based on a diverse set of unrelated complexity assumptions so that one technique will not break more than a handful of systems.

The most popular computational foundation for cryptographic security is hardness assumptions based on number-theoretic problems such as factoring, discrete logarithm, and elliptic logarithm [DH76, RSA78, BF03]. These problems are all related, so if one is broken it is likely that they all will be broken. (See [dM04] for further discussion of this.) Their security is not proven and is likely to either remain unproven or be broken. They are also vulnerable to quantum attacks, such as those pioneered by Shor [Sho94]. It is desirable to have many kinds of cryptographic primitives whose security is based on a wide array of unrelated assumptions, so that if one system is compromised, they are not necessarily all compromised. In this thesis, we present a system based on an alternative logic-based assumption that does not appear to be closely related to these other assumptions.

Specifically, we consider the use of the well known Boolean satisfiability prob-

lem to provide a very general cryptographic primitive, *secure set membership*. Secure set membership can be used to provide digital credentials, with or without identification, as well as for some signature problems such as timestamping. For example, consider a system for maintaining encrypted PINs for credit cards. Each credit card may have multiple PINs (for multiple users); any solution should hide the PINs in such a way that the system accepts valid PINs, but nobody can determine any valid PIN that he does not already know. If the system is used in a setting in which it is reasonable for the system to be able to determine which user it is talking to, then it is possible for the system to simply store hashes of all the valid PINs and compare a received hashed PIN with this list to determine if it is valid. This is an example of *credentials with identification*. However, if the users of the credit card do not want to identify themselves, or if the credit card issuer prefers to have users not identify themselves, except as a valid user of the credit card in question, when they make a purchase, then this results in the goal of *anonymous credentials*. For anonymous credentials, the user wants to prove that he has valid credentials without giving the credentials away.

In Chapter 4, we provide a means for constructing a secure set membership system that can be used both for credentials with identification and for anonymous credentials. Secure set membership can be used as an alternative to digital signatures for some applications including timestamping [BdM94]. We note that our system has the desirable property that each participant can choose her own set elements. In the setting of digital credentials, this allows participants to choose their credential values (rather than having them determined by a third party or as an output of a distributed credential generation algorithm), thus making the system suitable for use with credentials that are determined by user-chosen passwords or biometrics.

Our proposed solution is based on the Boolean satisfiability problem (SAT), which has not previously been used for digital credentials. The 3SAT problem is SAT with three terms per clause. Any SAT instance can be written as a 3SAT instance in polynomial time [GJ79]. We are aware that the use of the problem of finding witnesses for 3SAT instances as a security assumption is unusual and the practice of basing cryptographic hardness on *NP*-completeness is shaky in general, because the worst case hardness required for *NP*-completeness does not say anything about most cases or the expected case. However, we think it is of interest nonetheless. First, algorithmic advances and new computing models threaten many of the commonly used cryptographic assumptions, such as the hardness of factoring. Secondly, SAT is perhaps one of the most studied *NP*-complete problems, and a fair bit is known about how to choose instances that appear to be hard. We discuss this further in Section 4.4 and Chapter 5 in the context of our proposed solution. To introduce complexity, in Chapter 2 we discuss complexity results for cryptography. In Chapter 3 we give definitions that we will use later in this thesis. In Chapter 4 we present the system and discuss security and performance. In Chapter 5 we give an overall discussion of our system.

Our contribution includes three protocols with applications to anonymous credentials, credentials with identification, accounts with multiple users, and digital timestamping. Specifically, we provide a method for generating representations of sets of provided elements. We also provide a method of using a resulting representation to prove a particular element was in the set at the time the representation was generated and a method of using the representation to show a party holds a valid set element without revealing the element itself. Our representations are random 3SAT instances of a particular form that accepts the chosen witnesses. We show (Theorem 1) that among 3SAT instances that accept the selected witnesses

our algorithm can produce any 3SAT instance that has the selected number of clauses. The security of the scheme relies on the computational difficulty of finding satisfying assignments to such 3SAT instances. Our system has the following properties:

- It generates instances of 3SAT that are satisfied by a given set of strings.
- It generates any suitable instance of 3SAT. This is shown in Theorem 1.
- In combination with zero knowledge proofs for 3SAT, it provides interactive proofs that can be used for anonymous credentials.
- Assuming the 3SAT instances generated are appropriately hard, it provides security against an attacker either finding a participant's element from the information needed to verify set membership or finding other bit strings that satisfy the set membership problem.

There are several applications that motivate consideration of secure set membership. Some of these applications are not well addressed by our specific system due to limitations in the number of witnesses that can be supported, but they remain as a motivation to work on the secure set membership problem.

DIGITAL CREDENTIALS. Our system applies to anonymous credentials in a fairly straightforward manner. The credentials are the elements. They are generated using either the centralized protocol or the distributed protocol and they are verified using the proof of possession protocol. In this way, the credentials are all generated at once and then the instance is distributed to the verifiers. Verifiers use the instance to anonymously determine whether a member is credentialed. If credentials with identification are desired, the member can present his witness; the

verifier can then check that the witness satisfies the instance.

ACCOUNTS WITH MULTIPLE USERS. The system is also useful in situations where there need to be multiple authentication strings for a single account. An example is accounts with multiple users. Suppose there are three debit cards issued on one bank account and they all have the same number but each has a different PIN. The PINs can then be used as witnesses in constructing an instance. When a user wants to demonstrate that she is an authorized user of the account, she runs the proof of possession protocol using her PIN. This way, joint holders of an account can access the account without giving away their PINs (which might also be used for other accounts that are not shared even though this is not recommended practice).

Other applications of multiple user accounts include the use of RFID tags as witnesses in an access control system based on proximity sensors and other access control situations where it is not desirable to uniquely identify the user.

ACCESS CONTROL LISTS. On some multiuser servers, such as file servers, it may be desirable to have an access control list for a resource such as a file or a directory in a format that does not immediately yield the identities of the users who may access this resource. A straightforward encoding of the users' identities would yield to a dictionary attack. The user could, however, keep an access token which is only known to the user. The user could then use a set membership protocol to prove that he is in the access control list. The access control list could be extended by keeping more than one instance. The access control list instances could be generated in a variety of ways including by the authorized parties.

DOCUMENT TIMESTAMPING. Document timestamping [BdM91, BdM94] may require a little more explanation. In document timestamping applications, we think

of the distributed protocol as a distributed signature. All the parties participating in the protocol are attesting that one of their number knew each witness at the time the protocol was run by accepting the set that results from the protocol. It would not be possible for the protocol participant to execute the protocol and then choose a satisfying witness at some later date.

The timestamping system proceeds in rounds. All documents submitted during the same round are considered to be simultaneous, like patent applications arriving at the patent office on the same day. Each participant's witness is a hash of the document(s) she would like to timestamp. The distributed protocol is run, and everyone remembers the round's set, which is the timestamp. The parties may jointly publish it if they wish to allow anyone to verify a timestamp.

To verify that a document was submitted during a given round, the verifier merely needs to run the set membership protocol. The security of this system does not depend on computational security, in that if a cheating prover wishes to make his specific document appear to be timestamped and it does not satisfy the 3SAT instance, there is nothing he can do to change that. (We note, though, that in most practical settings, the adversary may be able to change his document in ways that do not affect its meaning, but do affect its encoding into a bit string, so this guarantee is not absolute [LdW05, WYY05b, WYY05a]).

Digital timestamping can be used for intellectual property disputes, among other applications. In the intellectual property application, a consortium generates a timestamp with each company using a hash of the hashes of all of its documents. Each company retains the daily timestamp and publishes it for other interested parties. In a patent dispute, for instance, a party can get all the other honest participants to attest to its possession of a document on or before a certain date. This could also be used to prevent backdating in stock or other business transactions.

A preliminary version of the work in Chapter 4 appeared at the Eighth International Conference on Information and Communications Security [dMW06]. We introduce the problem of secure set membership and provide a candidate solution for it. We also did experiments to both determine what parameters to use and to test our security assumptions.

Our contribution, in summary, is that we:

- Define the secure set-membership problem.
- Propose a method to solve the secure set membership problem using a trusted central authority.
- Propose a method to solve the secure set membership problem using a group of authorities who are not individually trusted.
- Experimentally determine recommended parameters of the system.
- Experimentally investigate the security of the system.

Chapter 2

Related Work

The set membership problem was first addressed by Benaloh and de Mare with one-way accumulators in 1993 [BdM94]. A number of schemes based on one-way accumulators were developed including schemes for digital credentials [CL02, BP97]. The schemes for credentials typically differ from the other schemes, which tend to concentrate on the idea of a distributed signature, in several ways. These include central authorities in the credential scheme, as well as support of additional properties such as revocation. All these schemes depend on the difficulty of the RSA problem for their security.

Another approach to set membership is to use Merkle trees or similar tree-based methods to store the elements of the set [Mer82, Mer88, BdM91]. In these methods, each participant retains a certificate and her own set element. In effect, each element of the set is signed by a central authority. However, these methods are either not storage-efficient or require more than a constant amount of time to check relative to the number of entries. For instance logarithmic time is optimal for Merkle trees [Szy04].

In a credential system, members of the credentialed group have, or are given,

credentials that they can use to prove their membership in the set of authorized persons, without revealing which of the members they are. Biometric data may be used to prevent transferability of credentials, together with zero knowledge proofs of knowledge, for a group member to prove to a verifier that she holds a valid credential without revealing it. Anonymous credentials have been widely studied and solutions based on various cryptographic assumptions have been given (e.g. [CL01, CL02, Acq03, IM03]).

Several approaches have been taken to digital credentials. Most of these approaches require a central authority (such as [BdM94, CL02]), though some approaches based on one-way accumulators do not require a central authority. Our approach can work with or without a central authority. The combination of one-way accumulators and zero-knowledge proofs was introduced by Camenisch and Lysyanskaya [CL02]. Other credential systems allow revocation of anonymity (e.g. Camenisch and Lysyanskaya [CL01]).

Our work makes use of the assumed computational difficulty of finding satisfying assignments to certain kinds of satisfiable 3SAT instances. A related use of the hardness of SAT for achieving security has been proposed for hiding information in anomaly detection applications [EAFH04, EFH04a, EFH04b, Esp05]. Their work is concerned with maintaining lists of information that, if compromised, will not compromise the larger system for applications such as intrusion detection. The central idea of our system is to represent an element of a set by an assignment to a set of variables, and the set of elements by a 3SAT instance that is satisfied by the corresponding assignments. In comparison, the work of Esponda et al. uses a SAT instance to represent a database; in their case, they represent the values *not* in the database by satisfying assignments.

We note that both our use of 3SAT and Esponda et al.'s use of SAT do not have

the same difficulties as earlier uses of *NP*-complete problems for cryptography, such as the knapsack problem (see [Odl90] for a nice survey of this history), because it is not necessary to embed a trapdoor to be used for operations such as decryption.

In this chapter, we explore what can be accomplished with complexity-based cryptography by first examining a result showing that one-way permutations that are *NP*-complete to invert are not likely to exist. In Subsection 2.2 we examine a method of determining the "usual" complexity of a problem.

We say that a cryptographic system is *complexity-based* if its security is derived from the belief that breaking it will lead to an efficient algorithm for a problem complete in a complexity class that is believed not to have efficient algorithms. If an algorithm can solve a problem complete in a complexity class, it can then solve any problem in that complexity class. In that spirit, when we say that a one-way function is *NP*-hard to invert, we mean that given an oracle that inverts it, a deterministic Turing Machine can use the oracle to decide an *NP*-hard problem in polynomial time. If inverting the function is *NP*-hard and can be written as a decision problem in *NP*, then we say that the function is *NP*-complete to invert.

In this section, we discuss limits to what can be accomplished with complexity-based cryptography. Subsection 2.1 shows that there exists a function computed by a polynomial-time bounded deterministic Turing machine with an oracle that inverts a one-way permutation, $P^{f^{-1}}$, where f is a one-way permutation, that is in $NP \cap \text{co}NP$ [Bra79]. Assuming, as is commonly believed, that $NP \neq \text{co}NP$, this means that a cipher cannot be *NP*-hard to invert. For this reason, in Chapter 4, we propose a function that is weaker than a one-way permutation and is strongly related to an *NP*-hard problem. This function is likely the function with the most applications based on complexity possible because of Brassard's theorem [Bra79].

This is because as functions require stronger assumptions, they are able to be applied to more applications. We also discuss another issue that arises with complexity-based cryptography. The issue is that while complexity classes are based on *worst-case* complexity, cryptography requires that all but an asymptotically negligible number of instances be hard.

2.1 One-way Permutations are Not *NP*-hard to Invert

We start with a 1979 result by Brassard that suggests that basing cryptography on *NP*-hardness may be futile [Bra79]. Specifically, Brassard shows that if any one-to-one one-way function whose input size is bounded on the size of the output is *NP*-hard to invert, then $NP = \text{co}NP$. It follows that one-way permutations cannot be shown *NP*-complete under the usual assumptions. For this result, the domain (which is a set of strings—i.e., a language) of the one-way function must be in *NP* and the range (also a language) of the one-way function must be in co*NP*. When the domain of the function is in *NP*; a Turing machine must be able to decide in nondeterministic polynomial time if an input is valid for the function. In most cases, where any string is a valid input, this is trivially true.

Brassard does not show this result directly, but rather shows a specific result for the discrete logarithm problem and sketches it for factoring, simply stating that it generalizes. It is worthwhile to take the time to understand how the generalization works because this is fundamental to understanding the computational complexity of breaking cryptographic systems. The domain of a one-way function is a binary string that can be interpreted as an integer. The decision problem is specified as follows: Given y, t such that $f(x) = y$, is $x > t$? Brassard's result is that this problem is in co*NP*. In order to show this, he must show that there is a

succinct witness that there is no x such that $x > t$. Since the function is one-to-one, it is sufficient to show that there exists an x' such that $f(x') = y$ and $x' \leq t$. This x' serves as a witness that there is no x such that $f(x) = y$ and $x > t$ by the pigeonhole principle. Therefore this problem is in co*NP*. Brassard goes on to show that if an *NP*-complete problem is in co*NP*, then *NP*=co*NP*.

Note that Brassard's result only applies to functions that are one-to-one. In contrast, Kozen showed a simple example of a one-way function that is not one-to-one but is *NP*-hard to invert [Koz06]. One-to-one one-way functions are known as *one-way permutations*. Impagliazzo and Rudich showed that if $P \neq NP$ then one-way permutations cannot be used as black boxes for stronger cryptographic primitives such as key agreement [Rud88, IR89]. One-way permutations trivially imply one-way functions as every one-way permutation is a one-way function. It remains open whether one-way permutations can be constructed from one-way functions. The above result that one-way permutations cannot be shown *NP*-complete under commonly held assumptions while one-way functions exist that can be shown *NP*-complete would tend to suggest that no such black-box construction exists.

Akavia et al. show that if a one-way function is *NP*-hard to invert in the average case, then co$NP \subseteq AM$ [AGGM06]. The complexity class *AM* is the class of all problems that have two-round interactive proof systems. It is generally believed that co*NP* is not a subset of *AM* so this is an even stronger result suggesting that it is not possible to construct a one-way function that is *NP*-hard to invert in the average case. This result was built on results by Feigenbaum and Fortnow [FF93] and Bogdanov and Trevisan [BT03]. Our system is weaker than a one-way function.

Although we cannot build a one-way function that is NP-hard on average to invert, Ajtai has discovered a relationship between average-case complexity and worst-case complexity for lattice problems that can be used to build cryptographic

primitives which are secure if the *LLL* algorithm is optimal or nearly optimal [Ajt04, Mic07]. This suggests that our approach to complexity-based cryptography can be made to succeed at least in some cases.

2.2 Generic Complexity

Complexity classes deal with worst-case complexity. In worst-case complexity, a problem is considered to be hard even if only a small number of its instances are hard. Another way of measuring complexity, called average-case complexity, is also not satisfactory as a system is not cryptographically strong if a significant minority of its instances are easy to break. A cryptographer wants to base a system on a problem for which breaking almost every instance is hard. This concept is captured by *generic complexity* [GMMU07]. A language $L \subseteq \Sigma^*$ is said to have generic complexity $O(f(n))$ if, for all but a negligible subset of Σ^n, L can be decided in $O(f(n))$ time for a given measure. The definition of generic complexity requires that a measure function be defined on the problem in order for "negligible" to be defined. There are different definitions of "negligible" depending on how strongly generic a function is, but generally it is based on taking the limit as the size goes to infinity. The usefulness of a generic complexity result depends on how well-chosen the measure is for the setting at hand.

A problem is said to be generically in a complexity class for a given measure if, except for a negligible density, the problem has an algorithm that fits the definition of an algorithm for that complexity class. This means that if a problem is generically polynomial time, there exists a polynomial-time algorithm which will solve all but a negligible number of instances of that problem. If a problem is generically *NP*, then there is a nondeterministic-polynomial-time algorithm for all but a

negligible number of instances of that problem.

Many problems that are hard in the worst-case and even some that are undecidable have generically polynomial time algorithms relative to some natural measure. This means that it is not enough for cryptographic security to prove the complexity of breaking a system in the worst-case, security based on complexity must also be shown to be hard generically, or at least not be generically polynomial time, in order for the security to be derived from the complexity class. Since it is not known how to prove a cryptosystem secure if problems in a complexity class are difficult, this additional requirement makes it even more unlikely that it will be possible to prove a cipher difficult to break based on complexity.

An example of a one-way function that is *NP*-complete to invert, but is generically easy to invert, is the function used by Kozen [Koz06]. This function takes a Boolean formula and a set of assignments as inputs then outputs the Boolean formula and the result of the evaluation of the formula using the assignments as outputs. This function is *NP*-complete to invert because it is possible to convert any instance of SAT to a value in the range of the function and then read a witness from the input. However, for any natural measure on all but a negligible number of outputs, finding an input that makes the output true is easy [GMMU07].

Chapter 3

Definitions and Models of Computation

This chapter contains many of the definitions used in this thesis including definitions related to the models of computation used to determine complexity or expected complexity. The definitions in this chapter that are defined in other works are presented with sufficient detail for their use in this thesis; where more detailed definitions exist, they may be obtained from the cited sources.

3.1 Turing Machines

A *deterministic Turing machine* is a model of computation described in texts such as [HMU01, Pap95, Koz06, DSW94] that consists of an alphabet Σ, a tape, a set of states, a transition function, and a tape alphabet which includes a space symbol. The transition function shows how to change configurations by taking a symbol (under the tape read/write head) and a state to return a new symbol (to be written), a new state, and a direction to move the read/write head. A *nondeterministic*

Turing machine is like a deterministic Turing machine except that there is a transition relation rather than a function. A deterministic Turing machine is in a single configuration at any given time while a nondeterministic Turing machine may be in multiple configurations simultaneously. These multiple configurations can alternatively be described as choosing from among multiple successor states and always choosing "correctly". Starting from a configuration of a nondeterministic Turing machine and selecting only one subsequent configuration from the possible subsequent configurations from the transition relation, and continuing to do this recursively, yields a *path of computation*. A deterministic Turing Machine is said to *accept* its input if it halts in an accepting state while a nondeterministic Turing Machine is said to *accept* its input if at least one path of computation halts in an accepting state. Likewise a deterministic Turing machine *rejects* its input if it halts in a rejecting state while a nondeterministic Turing Machine *rejects* its input if all paths of computation halt in a rejecting state.

Alternating Turing machines may be found in texts such as [Koz06]. An *alternating Turing machine* uses a relation rather than a function for the transition function like a nondeterministic Turing machine. This results in a tree of configurations for an input. States in an alternating Turing machine may be labeled with a $\neg$, a $\vee$ or a $\wedge$ so that configurations may alternate between the closest higher configuration on the tree being a $\vee$ and a $\wedge$. A $\neg$ label indicates that the value of children should be reversed. There may be a finite number of alternations. A $\wedge$ means that all computation paths descending from the current configuration must accept for the path to accept and a $\vee$ means that at least one computation path descending from the current configuration must accept in order for the path to accept.

A *probabilistic Turing machine* is a Turing machine with an additional tape of random bits that may be read sequentially and not written [Koz06]. The resulting

computation may be a function of the random bits as well as the input. If two probabilistic Turing machines have the same random bits, same states, and same transition function we call them *random clones*.

An *oracle Turing machine* as defined in [Koz06] is a Turing machine with an oracle tape. The oracle Turing machine may write strings to the oracle tape and enter a special state where the oracle replies by writing a single symbol to the oracle tape which may be read.

A *language* $L \subseteq \Sigma^*$ is a (possibly infinite) set of strings written with the alphabet Σ. A *class* is a (possibly infinite) set of languages [HMU01, DSW94]. A language $L \subseteq \Sigma^*$ *is decided* by a Turing machine, M, if for any $x \in L$ as input, M halts in an accepting state and for any $x' \notin L$, M halts in a rejecting state [HMU01, DSW94]. Turing machine A *simulates* Turing machine B if A produces the same output as B [Lyn96].

3.2 Complexity Classes

Information on complexity (including *NP*-completeness) is available in Garey and Johnson's 1979 book [GJ79] as well as more modern texts[1] [Pap95, Koz06, Koz92].

Recall that a *class* is a (possibly infinite) set of languages [HMU01, DSW94]. A *complexity class* is a class whose membership is determined by the existence of a Turing machine or an instance of another specified model of computation with some specified property or resource bound that can decide languages within the class. Many complexity classes, such as P and NP, are defined by the model of

[1] [GJ79] contains reductions for many *NP*-complete problems. [Pap95] focuses on *NP*. [Koz06] discusses many complexity classes and prerequisites may be found in [Koz92], a quarter of which is about complexity classes.

computation required to solve it in polynomial time.

Given a language $L \subseteq \Sigma^*$ and a set of valid instances: $U \subseteq \Sigma^*$, the complement of L is $\overline{L} \subseteq U$ such that:

$$x \in L \Leftrightarrow x \notin \overline{L}$$

Given a class C, co$C$ is the set of languages which are complements of languages in C [Koz06].

A function $f : \mathbb{N} \mapsto \mathbb{N}$ is said to be *polynomially bounded* if there exist $k, c \in \mathbb{N}$ such that $n > c \Rightarrow f(n) < n^k$ [Koz06]. The complexity classes P and NP can be found in many texts [GJ79, Pap95, Koz06]. The complexity class P is the class of languages which can be decided by a deterministic Turing machine in a polynomially-bounded number of configurations. The complexity class NP is the class of languages which can be decided by a nondeterministic Turing machine with each computation path having a polynomially-bounded number of configurations.

The complexity class Σ_k^P is the class of languages which can be decided by an alternating Turing machine with each computation path having a polynomial number of configurations starting with a $\vee$ and having at most k alternations. Similarly, Π_k^P is the class of languages that can be decided by an alternating Turing Machine starting with a $\wedge$ and having at most k alternations. $NP = \Sigma_1^P$ and co$NP = \Pi_1^P$. The *polynomial hierarchy*, *PH*, is the class of languages which are accepted by Σ_k^P or Π_k^P for some k [Koz06]. *PSPACE* is the class of languages that can be decided by a Turing Machine using a polynomially-bounded on the size of the input amount of space on its tape [Koz06].

Logspace and polynomial reducibility are defined in many texts [GJ79, Pap95, Koz06]. Given a language $A \subseteq \Sigma^*$ and a language $B \subseteq \Delta^*$, A is logspace re-

ducible to B, denoted $A \leq_M^{\log} B$, if there exists a function $\sigma : \Sigma^* \mapsto \Delta^*$ which can be computed by a deterministic Turing machine using $\lceil \log_2 n + 1 \rceil$ space for n input symbols. Similarly A is polynomially-reducible to B and written $A \leq_M^P B$ if σ can be computed in a polynomially-bounded number of configurations on a deterministic Turing machine.

The idea of completeness is discussed in many standard texts [GJ79, Pap95, Koz92, Koz06]. A language L is *complete* for a complexity class C, denoted C-complete, if $X \in C \Leftrightarrow X \leq_M^{\log} L$ and $L \in C$. If every language in C is reducible to L (but L may not be in C) then L is said to be C-hard. Likewise we say that a language L is *polynomially complete* for a complexity class C if $L \in C$ and $X \in C \Leftrightarrow X \leq_M^P L$. Note that all languages that are C-complete are C-hard.

The *counting problem* on a language is the integer valued function that takes an instance and returns the number of witnesses that make it true. For any integer valued function whose range can be bounded by a function $g : \Sigma^* \mapsto \mathbb{N}$, computable in a polynomially-bounded number of configurations, $f : \Sigma^* \mapsto \mathbb{N}$, there is an oracle $\geq_f$ that takes as its input $x\#t$ and writes "yes" if $f(x) \geq t$ or "no" if $f(x) < t$. Using binary search [Knu98, AHU74, CLRS01], a Turing Machine with the oracle $\geq_f$ can output the value $f(x)$ in $\lceil \log_2 g(x) + 1 \rceil$ oracle queries. The complexity class *#P* is the class of languages decidable in a polynomially-bounded number of configurations of a deterministic Turing Machine relative to an oracle $\geq_{\#L}$ where $\#L$ is the number of witnesses for an instance of $L \in NP$ [Koz06]. For example, the counting problem on SAT is the number of truth assignments that satisfy the Boolean expression.

3.3 Oracle Notation

The class C^O of languages is defined to be the set of languages that are in the class C using Turing machines with access to an oracle O. This is called *C relative to O*. If O is a language, then we are considering an oracle that decides that language; if O is a class of languages, then it is an oracle that decides every language in the class, i.e., a language complete for the class [Koz06, DSW94].

3.4 Zero Knowledge

A protocol for proving a proposition is *zero knowledge* if the verifier, with no more information than the proposition to be demonstrated, can produce transcripts identically distributed to those that are produced by the interaction between the prover and the verifier [MvOV97].

3.5 Set Membership

In this section, we define secure set membership systems. A *secure set membership system* consists of two protocols. First, the set must be established. Later, holders of set elements can prove their elements' set membership to others. Depending on the application, it may be desirable for the proof to reveal the set element or to keep it secret. Specifically, we have the following definitions.

Definition 1. A *set establishment protocol* is a protocol carried out by some number m of *participants* $P_1, \ldots, P_m$. Each P_i holds as an input *set element* w_i. The output of the protocol is a *set representation* $T = T(w_1, \ldots, w_m)$. If the system is randomized, then each participant also gets a string of random bits. ■

In our system, the set establishment protocol also takes a string of random bits that guide its nondeterministic choices. Thus the function looks like: $T = T(r, w_1, \ldots, w_m)$ where r is a string of random bits. These random bits are available to all participants and may be public.

Definition 2. A *set membership protocol* is a protocol carried out by a participant P holding a set element w and a *verifier* V holding a set representation T. An honest verifier *accepts* if and only if the representation T was generated from a set of elements including w, even if P is cheating. The verifier may learn w. ■

Obviously, the set membership protocol is unsuitable for credential systems in which the set elements are reusable credentials, because it allows both V (and possibly eavesdroppers) to learn w and thereby to masquerade as P in the future to others. The protocol is also unsuitable for anonymous credential systems unless further measures are taken, because it allows V to distinguish between different provers because they have differing credentials. Fortunately, both of these difficulties can be eliminated by using a proof of possession protocol, defined below, instead of a set membership protocol.

Definition 3. A *proof of possession protocol* is a protocol carried out by a participant P holding a set element w and a verifier V holding a set representation T. An honest verifier accepts if and only if the representation T was generated from a set of elements including w, even if P is cheating. The verifier V does not learn w, even if V is cheating. The verifier, with no information other than the set representation and the knowledge that the prover has a valid element of the set, can simulate the prover, generating the same transcript as would be generated by the valid interaction between the prover and verifier. This property is called *zero knowledge* (see Section 3.4). ■

In Definition 3, we assume all participants are computationally bounded. In our system, the elements of the set are interpreted as assignments to a set of variables. These assignments are called *witnesses*, because in our proposed solutions they are witnesses to the satisfiability of 3SAT expressions. Our solutions depend on the computational infeasibility of finding witnesses for certain 3SAT expressions (also called *instances*). We discuss the validity of this assumption further in Section 4.4.

When we discuss a 3SAT instance, we pay attention to two parameters. These are the number ℓ of variables and the number n of clauses, which is also called the *size* of the instance. We also consider the *clause density* $\alpha = \frac{n}{\ell}$, which is an important parameter for determining the difficulty of a 3SAT instance [ABS03]. We refer to a 3SAT instance that represents the set of elements as a *set representation* or, when clear from context, simply as a *set*.

Chapter 4

Secure Set Membership Using 3SAT

In this chapter, we describe our secure set membership protocols. We first describe in Section 4.1 a centralized process for a trusted party to establish a set representation for a set of given elements. In Section 4.2, we describe a distributed version of the set establishment protocol, which can be carried out by the participants holding set elements and does not require a centralized trusted party. In Section 4.3, we describe how to show set membership for elements of the established set. We discuss the security of our solutions in Section 4.4.

4.1 Centralized Set Establishment Protocol

Let $W = \{w_1, w_2, \cdots, w_m\}$ be a set of assignments to ℓ variables, $\{v_1, \ldots, v_\ell\}$. Each w_i represents an individual element called a *witness*. The trusted party, say $\mathcal{T}$, generates a set representation for W—that is, a 3SAT instance satisfied by each $w_i \in W$. To do this, $\mathcal{T}$ repeatedly generates random clauses that are the conjunction of 3 literals over variables in V. $\mathcal{T}$ checks each clause generated to determine whether it is satisfied by every $w_i \in W$. If there is some $w_i \in W$ that does not satisfy

the clause, then $\mathcal{T}$ discards the clause and randomly selects a replacement clause which goes through the same test. Once n satisfied clauses are found, where n is a security parameter representing the desired size of the expression, their conjunction forms the desired set representation T, which is output by $\mathcal{T}$. The complete algorithm is given in Algorithm 1.

Note that the output T is an instance of the 3SAT problem satisfied by the assignments that the participants have specified as elements. It may also be satisfied by some other unknown assignments. However, even if there are such spurious witnesses, that does *not mean* they are easy for an attacker to find. Nonetheless, it seems desirable to avoid having many such spurious witnesses. One can reduce the number of spurious witnesses by choosing a large n, because the probability of a given assignment satisfying a 3SAT instance decreases exponentially with the size of the instance. Specifically, n should be chosen to be large enough to satisfy three security criteria:

- The conjunction of the clauses should be satisfied by very few assignments that are not valid elements.
- The size of the conjunctive normal form (CNF) expression that is made by the clauses should be large enough that there is high probability that it is not an instance of SAT for which an efficient solution is known.
- The size of the CNF expression should be large enough that it can potentially be computationally infeasible to find satisfying assignments.

In general, this can be accomplished by choosing a suitably large number of variables and setting the clause density to a suitable value. The value for m is a parameter for the system. The value ℓ is fixed based on security requirements and the

clause density is chosen to be something believed to be hard. The security of the scheme is discussed further in Section 4.4.

Input: A set of variable assignments $W = \{w_1, w_2, \cdots, w_m\}$ to ℓ variables and the target number n of clauses.
Output: A 3SAT instance satisfied by all $w \in W$.

While there are fewer than n clauses do:

1. Select three different random numbers $\{i_1, i_2, i_3\} \in \{1, \ldots, \ell\}$.
2. Select three random bits n_1, n_2, n_3. For each bit n_j, if $n_j = 0$, the literal v_{i_j} is added to the clause; if $n_j = 1$, the literal $\neg v_{i_j}$ is added to the clause instead.
3. If another clause has the same three variables and corresponding negations and return to Step 1.
4. For each w_j do

   If, for all $i \in \{i_1, i_2, i_3\}$ (($n_i = 1$ and v_i is set in w_j) or ($n_i = 0$ and v_i is not set in w_j)) then goto Step 1.
5. Add the clause represented by $\{(n_1, v_{i_1}), (n_2, v_{i_2}), (n_3, v_{i_3})\}$ to the instance.

Algorithm 1: A centralized protocol for establishing a set

We now turn our attention to the computational complexity of this algorithm. We note that there is some chance that the algorithm might not even terminate, if there are not a sufficient number of available clauses that satisfy the given witnesses. However, if ℓ is chosen relatively large in comparison to m, and ℓ is sufficiently large compared to m and n, there should be a sufficient number of clauses that satisfy the witnesses. In order to make it possible for the algorithm to terminate, we need the following condition to hold:

$$n \leq \left\lfloor 8\frac{\ell!}{(\ell-3)!}((7/8)^m) \right\rfloor$$

The right hand side of the equation is the total number of possible clauses multi-

plied by the fraction of the clauses that will be accepted. This gives the number of clauses that may be included in the final instance. It is desirable to choose ℓ based on security requirements. Finding witnesses is conjectured to be exponential in ℓ for at least some instances [GJ79]. A discussion of how to choose n follows.

Assuming a large number of clauses satisfy the given witnesses, consider a particular witness representing one set element and consider a single randomly chosen 3SAT clause. There are three variables in a clause, all of which are given some assignment in the witness. Each variable in the clause can appear as a literal in either positive or negative form, so there are eight possible cases. Of these, seven are satisfied by the witness; it is only not satisfied (and therefore not accepted) in the case where none of the three literals are satisfied. Thus, the probability of a clause satisfying one witness is $\frac{7}{8}$. If there are m witnesses, then the probability of a clause satisfying all of them is $(\frac{7}{8})^m$. It follows that the expected number of tries required to generate a clause in the set representation is $(\frac{8}{7})^m$. It takes $O(\log \ell)$ bits to represent a clause and the clause must be checked against m witnesses, each of which can be done in constant time. Therefore, it takes $O(m \log \ell)$ time to test a clause to determine whether it is satisfied by all the witnesses.

In order to generate n clauses, it is necessary to find n distinct clauses that are satisfied by W. As each clause is found, it becomes slightly harder to find the next clause, as duplicates will sometimes be chosen. However, as long as n is very small relative to the total number of clauses that satisfy W, this has a negligible effect. If the probability that a random chosen clause passes both tests (satisfies W and is not a duplicate) were fixed at $(\frac{7}{8})^m$, then the expected running time to generate a set representation would be $O(n((\frac{8}{7})^m)m \log \ell)$. We note that in cases where n is a significant fraction of the total number of clauses that satisfy W, then this would not be the case.

In practice, this means that it is only computationally efficient to generate an instance for at most up to around a hundred witnesses. A hundred witnesses leads to an expectation of 629,788 rejected clauses per accepted clause, easily doable with current computers. When the number of witnesses reaches a hundred and fifty, there is an expectation of about five hundred million rejected clauses for each accepted clause, probably infeasible for a typical modern computer for even a small number of clauses.

4.2 Distributed Set Establishment Protocol

We now discuss the distributed protocol for establishing T, which is given in Algorithm 2. This algorithm works for *honest-but-curious* participants, who are assumed to follow their specified protocols but who may pool their information to try to learn more than they are supposed to. It also has some resilience against cheating participants who do not follow the protocol; for example, cheating parties can cause an easy instance of 3SAT to be chosen, but in some cases the other participants can detect that this may be happening. We will discuss security further in Section 4.4. At a high level, the protocol executes as follows: the participants locally generate local copies of the same random clause. Each determines if the clause is satisfied by her own witness and communicates this information to the others. If the clause is satisfied by all the witnesses, it is kept. Otherwise, it is discarded.

In order to protect the participants' witnesses from being disclosed, we use a verifiable secret-ballot election scheme by Benaloh [Ben87]. The scheme is based on *homomorphic encryption* and *secret sharing*. It operates by designating some participants as *tellers*. Participants give secret shares of their votes to the tellers. The

Input: A set of variable assignments $W = \{w_1, w_2, \cdots, w_m\}$. Each w_i is known to participant P_i. All participants also know the number ℓ of variables to be used and the target number n of clauses, as well as a sufficiently long random string R.
Output: An instance of 3SAT that is satisfied by all participants' witnesses.

- set $i := 0$,

$$maxrejectbase := \frac{\left(\frac{8}{7}\right)^m}{8} \cdot \frac{S + \log n}{8 \log \frac{8}{7}}$$

$$clauses := \left\lfloor \frac{8\ell(\ell-1)(\ell-2)}{\left(\frac{8}{7}\right)^m} \right\rfloor$$

- While there are fewer than n clauses do:
  1. $maxreject := maxrejectbase \frac{clauses}{(clauses - i)}$, $i := i + 1$
  2. Using R, select three different variables i_1, i_2, i_3 and three flags n_1, n_2, n_3.
  3. Construct the clause where the flags denote the negation of variables.
  4. If the clause is equivalent to a clause already generated, discard it and return to Step 2.
  5. Hold a verifiable secret-ballot election (see [Ben87]) using "yes" if the clause is satisfied by the witness and "no" otherwise. If the tally is unanimously "yes", then add the clause to the instance. Otherwise, delete it. Each teller should maintain a running sum of each participant's shares of votes.
  6. return to Step 2.
- Use the homomorphic property to compute the number of "no" votes for each participant. If one or more exceeds *maxreject*, discard all the clauses.

Algorithm 2: A distributed algorithm

tellers then use the homomorphic properties of the secret-sharing scheme to compute shares of the tally. They then collaborate to compute the actual tally and provide a proof to the participants that the tally was computed correctly.

In order to detect cheating of individual participants in our scheme, the tellers count the number of times that any participant votes "no" for any given clause. This can be accomplished without revealing the votes to the tellers by using the homomorphic property of the election scheme. The tellers maintain a running sum of

each participant's votes and collaborate to determine that sum after a clause is chosen. If this sum exceeds a threshold value *maxreject*, then the instance is discarded and the protocol restarted from the beginning. The *maxreject* formula requires a security parameter; we suggest $S = 18$ or $S = 19$ for our recommended parameters. Depending on the application setting for the protocol, it may be desirable to exclude participants who have exceeded the *maxreject* threshold some number of times from further participation. We note that even if a cheating participant succeeds in influencing the outcome of the protocol, she can neither learn another participant's witness nor cause another participant's witness to not satisfy the resulting 3SAT instance.

The goal is to choose *maxreject* high enough so that it detects cheating at levels that could lead to malicious participants being able to break the security of the result, but low enough so that it does not unnecessarily restart the protocol when no participants are cheating.

In order to heuristically determine a good value for *maxreject* for a given round, we need to know what the total number of possible clauses is. This is:

$$clauses = \left\lfloor \frac{\ell!/(\ell-3)!}{(8/7)^m} \right\rfloor$$

Define i as the current round and S as a security parameter. As a threshold, we suggest:

$$maxreject = \left(\frac{\left(\frac{8}{7}\right)^m}{8} \cdot \frac{(S + \log n)}{8 \log \frac{8}{7}} \right) \frac{clauses}{(clauses - i)}$$

which is derived as follows. As mentioned previously, the probability of a random clause satisfying a given witness is $\frac{7}{8}$. The first term of the formula for *maxreject* is the inverse of the probability of all the witnesses being satisfied for a single clause

divided by the number of them that a single witness rejects. This is not sufficient to give a useful probability of an honest run not being rejected because there are n clauses yielding a probability that all n are satisfied of 2^{-n}. We multiply by the logarithm of the number of witnesses converted to logarithm base $\frac{8}{7}$ and divide by 8 (to convert from elections to rejections) in order to bring the probability up to one half that all the clauses will be accepted. A security parameter is added in this process to bring the probability to the desired level. It is necessary to use a higher security parameter with fewer witnesses. Multiplying by the ratio between the total possible clauses and the remaining possible clauses scales the total number of clauses that may be considered to account for clauses that have already been chosen. If $i = clauses$ then there are no more clauses to add and the ratio cannot be computed.

This choice of *maxreject* experimentally seems to be a good choice. See Figures 1 and 2. These figures show the relationship between the actual number of clauses rejected and the value from the *maxreject* formula. You will note that the actual number of tries is below *maxreject* but that some of them approach *maxreject*. The graph 1 also shows the curve for *maxreject*. This curve grows as the possible clauses get used up. The graph 2 plots the actual number of tries for each clause. These vary considerably, but tend to stay under a certain curve. We tried to capture this curve with *maxreject*.

We recommend a value of S between 18 and 21. The larger values should be used with a smaller number of witnesses. These recommendations are based on experiments using our recommended parameters.

To provide termination and also to provide some protection against multiple cheating participants colluding and "spreading out" their "no" votes in order not to individually exceed the *maxreject* threshold, it would also be a good idea to have

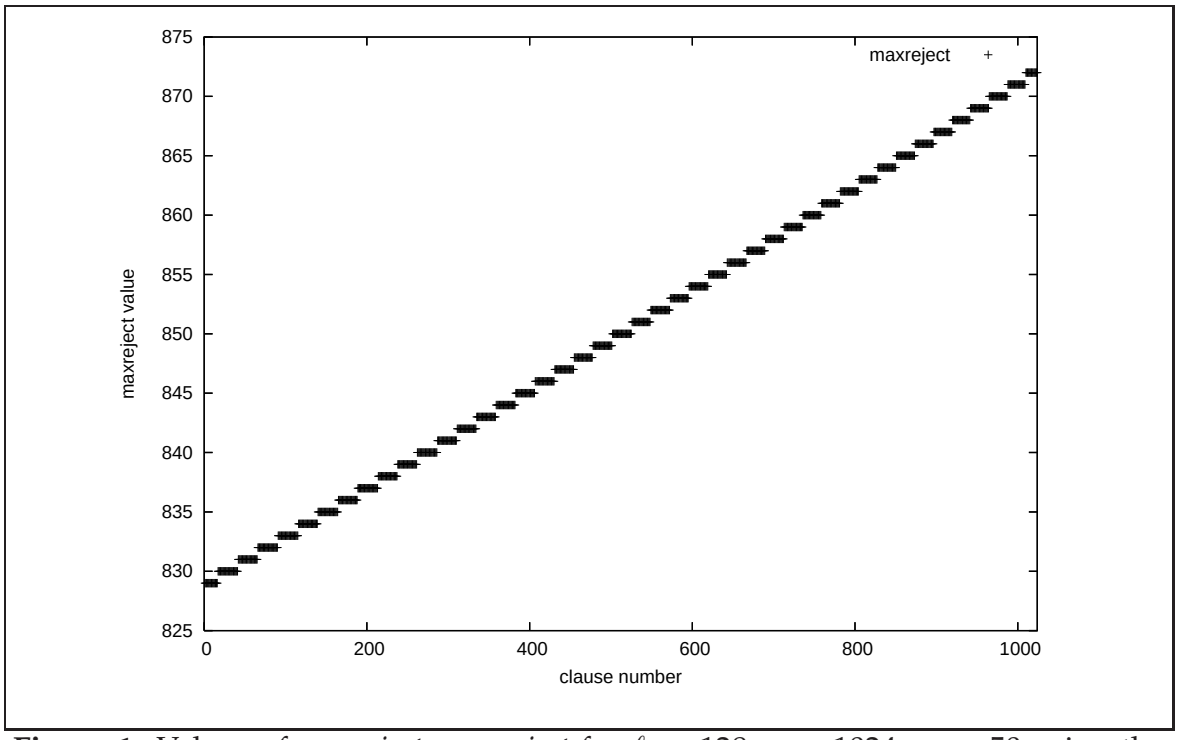

Figure 1: Values of *maxreject*: *maxreject* for $\ell = 128, n = 1024, m = 50$ using the formula in Algorithm 2.

a check in each iteration of the while loop that the loop has not been executed too many times, and to abort the protocol if this occurs.

In our set establishment protocol, the participants have a public shared source R of random or pseudorandom numbers. A theoretical model of algorithms executing with shared random information can be obtained with our notion of random clones in Section 3.1, or with the concept of random beacons [Rab83, Ben87]. If there are at least two witnesses, then R may be public. Using R, each participant generates a clause as the disjunction of three elements. Since the same random source is used, all the participants generate the same clause. The participants hold a verifiable secret-ballot election. If the tally is unanimously "yes", the clause is kept; otherwise, it is rejected. If any participant votes "no", then the clause is discarded. This process is repeated until the target number n of clauses has been

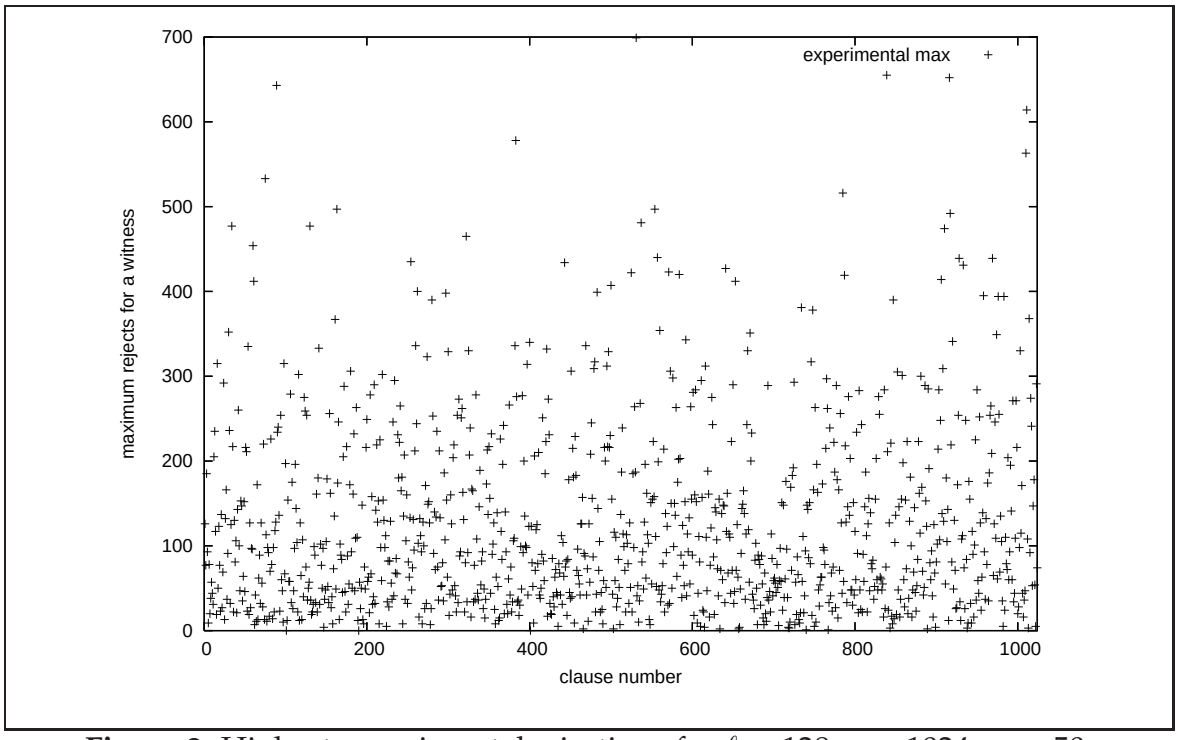

Figure 2: Highest experimental rejections for $\ell = 128, n = 1024, m = 50$.

generated.

It is easy to verify that the output T is satisfied by all the inputs $w_1, \ldots, w_m$, so Algorithm 2 meets the definition of a set establishment protocol. Assuming that parties behave honestly, the expected number of tries to find a clause is $(\frac{8}{7})^m$ as in the centralized protocol of Section 4.1.

4.3 Set Membership

Our set representations lend themselves easily to both set membership and proof of possession protocols.

Set membership involves a participant P, who knows his element w, and a verifier V, who knows T. P wants to convince V that T was generated as a set representation including the element w. In our case, then, P wants to convince V

that w satisfies T.

A straightforward set membership protocol (in which V is allowed to communicate w to P, as per the definitions in Chapter 3), is for P to communicate w to V, who can then easily check in polynomial time whether w satisfies T. If it does, V accepts; otherwise, V rejects.

For proof of possession, it is important that the verifier never learns the credentials and cannot impersonate the prover. Fortunately, in our solution, it is not necessary to present the element to show set membership, but rather it is sufficient to prove that one knows a satisfying string. This can be done without revealing the witness via a zero knowledge proof. There are simple zero knowledge proofs for 3SAT satisfiability that rely on factoring [BC87, Ben87]. Zero knowledge proofs are also possible using trapdoor one-way functions through a generic construction that applies to any *NP*-complete problem [GMW87]. Furthermore, this can be made secure against quantum computers [Wat06], in keeping with our motivation to avoid reliance on number-theoretic assumptions.

There are a number of parameters that must be set for our algorithms to work. These include the number of witnesses, ℓ, the clause density, α, and from these the number of clauses, $n = \alpha\ell$. These parameters affect the number of witnesses the instance may hold, the efficiency of the system, and the security of the system. The parameters generally need to be chosen with the security of the system in mind.

One potential problem which must be addressed when choosing parameters is spurious witnesses. A spurious witness is a string that satisfies the instance but was not a chosen witness. A random spurious witness does not compromise the system, because it is as hard to find as a legitimate witness. A spurious witness a short Hamming distance from a chosen witness may compromise the system using our security definitions because it is computationally feasible for a participant

with the chosen witness to find. We call these witnesses short-Hamming-distance witnesses. Using alternate security definitions that require the prover to be a participant, or authorized by a participant, rather than that the string be part of the set establishment protocol eliminates short-Hamming-distance attacks as a security issue because in order to conduct such an attack, one must know a chosen witness. Which set of security definitions should be used depends on the application. To be as general as possible, we carry out our analysis with the strict definitions presented in Section 3.5.

The number ℓ of variables is a key parameter for determining the size of the instance that will be generated by the algorithm. Experimentally, we have determined that ℓ should be at least $8,192$ but preferably $16,384$. The larger ℓ is, the larger the resulting instance is. Another parameter that needs to be set is α. We did experiments to determine these parameters and constraints on the system. These experiments are described in Section 4.4.

Experimentally, we have found that $\alpha \geq 64$ is necessary to prevent a SAT solver called MiniSat (see page 38) from deciding the instance. We have also found that if α becomes too large, then too much information is given, which also allows MiniSat to decide the instance. A larger α does provide additional protection against the existence of spurious witnesses a short Hamming distance from a chosen witness. An estimate of the number of instances with spurious witnesses a short Hamming distance away has been calculated and experimentally verified (see Section 4.4.2 on page 44) to be: $O(e^{-3\alpha/7})$. For up to five witnesses, the number of spurious witnesses a short Hamming distance away was experimentally determined to be negligible with $\alpha = 64$. We recommend using $\ell = 16,384$ and $\alpha = 64$.

4.4 Security

The security of the set membership protocol and the proof of possession protocol depends on the difficulty of finding witnesses that satisfy a set representation T constructed by the set establishment protocol. We show below in Theorem 1 that the representation T can be any of the instances of 3SAT with n clauses and ℓ variables satisfied by the specified assignments $W = \{w_1, \ldots, w_m\}$. The instance T may possibly be satisfied by some other assignments. The probability that T is hard is the same as the probability that it is hard to find a witness for a such an instance of 3SAT. Unfortunately, it is not known what this probability is. (In fact, if $P = NP$, then the probability is zero.)

Our system rests on the assumption that a sufficiently large random instance of 3SAT satisfying a given set of witnesses and having an appropriately chosen clause density has a high probability of being hard to solve. If this assumption holds, then it is hard for anyone to find a witness that is not an element. It is also hard for a party who does not already know an element of T to find one. These two properties provide the security for both the set membership protocol and the proof of possession protocol. In particular, for the set membership protocol, the ability of an adversary to succeed in forging a witness without overhearing one is precisely the adversary's ability to determine a satisfying assignment to T, because this property can be exactly checked by the verifier. In the case of the proof of possession protocol, the security additionally relies on the soundness of the zero knowledge proof. An adversary who cannot find a valid witness has only negligible probability of convincing the verifier to accept.

Theorem 1. *Algorithm 1 can generate any* 3SAT *instance consisting of n different clauses that is satisfied by all the assignments in W. Algorithm 2 can also.*

Proof: The same argument applies to both Algorithm 1 and Algorithm 2, because both algorithms save or reject clauses for the same reasons.

Consider the "random algorithm," which simply has a list of all the possible instances consisting of n distinct 3-literal clauses over ℓ variables that are satisfied by all $w \in W$ and selects one instance uniformly at random.

First, we show that our algorithm generates the same set of instances as the random algorithm. Suppose a possible 3SAT instance (in the random algorithm's list) cannot be generated by our algorithm. Then a clause in it must be rejected by our algorithm either because it is a duplicate or because some assignment does not satisfy the clause. It cannot be a duplicate, as this violates the requirement for the random algorithm's list that the clauses be distinct. If some assignment does not satisfy the clause, then no instance including that clause is satisfied by the assignment. Therefore, including it would violate the condition for the random algorithm's list that it must be satisfied by W. Hence, all instances in the list drawn on by the random algorithm are candidates for generation by our algorithm.

Conversely, suppose a 3SAT instance generated by our algorithm cannot be generated by the random algorithm. Then there are two possible reasons. The first is that there is a duplicate clause resulting in the number of unique clauses being less than n. This instance cannot be generated by our algorithm because the duplicate clause will be suppressed. The other possible reason is that it is not satisfied by one of the witnesses. In this case, one of the clauses is not satisfied by that witness (as the instance is a conjunction of the clauses). This clause will be rejected by our algorithm, so this instance cannot be generated. Therefore, the set of instances selected by the random algorithm is exactly the set of instances that our algorithm can generate. ■

Theorem 1 states that, given ℓ and n, the system can generate any 3SAT instance of ℓ variables with n clauses that is satisfied by the specified witnesses. We make some observations and propose some heuristic recommendations for selecting the security parameters as follows.

Beyond a certain threshold, increasing the number of variables without increasing the number of clauses actually reduces security because there are not enough instantiations of the variables to make the problem difficult. This is why we focus on the clause density rather than the number of clauses.

Recall that the clause density of an instance is defined as $\alpha = \frac{n}{\ell}$. Alekhnovich and Ben-Sasson [ABS03] show that if $\alpha \leq 1.63$, then the instance can be solved in linear time. They also demonstrate empirically that $\alpha < 2.5$ seems to be easy to solve. We recommend taking $\alpha \geq 64$ (i.e., choosing $n \geq 64\ell$) for security. For example, $\ell = 16,384$ and $n = 1,048,576$. The size of the problem was selected to be larger than the typical size of SAT-Race benchmarks and to correspond with our suggested clause density [HS00, Sin06]. We carried out experiments with SAT solvers to determine if problems of this size are easily solvable. We tested these values experimentally for up to five witnesses and neither the MiniSat solver (see page 38) nor short-Hamming-distance attacks could solve the instances.

The *phase boundary* of 3SAT is the clause density at which instances go abruptly from being mostly satisfiable to mostly unsatisfiable. The 3SAT decision problem—determining whether a 3SAT instance is satisfiable or not—is believed to be hardest when instances are just above the phase boundary [HS00]. However, our problem is a little different. Our set representation instances are always satisfiable (since they are specifically chosen to satisfy a particular set of witnesses). The problem at hand for an attacker is to find a satisfying assignment. We conjecture that the problem of finding satisfying assignments for instances that are known to be sat-

isfiable gets harder as the probability of a random instance of the same parameter being satisfiable gets smaller—i.e., well above the phase boundary.

A certain number of variables are trivial in any particular instance (i.e., because they either do not appear in positive form or in negative form, and therefore it is clear how to set them in a satisfying assignment). This can reduce the security of the system, by making it easier for an adversary to find satisfying assignments. Additionally, once the trivial variables are assigned, an adversary can then remove those clauses from consideration, potentially resulting in more trivial variables.

If our instances were random among all 3SAT instances with n clauses and ℓ variables, then the expected number of trivial variables could be limited by taking the clause density sufficiently large. However, as noted before, our instances are random only among those 3SAT instances that are actually satisfied by the set W of witnesses. Experimental results suggest that these instances are difficult for a small number of witnesses but become easier at an exponential rate as the number of witnesses is increased. This is mitigated by having a large number of variables (16,384). Still, with the exponential growth rate of trivial variables, it is not safe to have more than ten witnesses.

SATLIB contains resources for experimental research on SAT and 3SAT, including the results of competitions in solving random SAT instances [HS00]. In addition, there is an annual competition to create SAT solvers called SAT-Race [Sin06]. To date, not much progress has been made on high clause-density instances. We did our experiments using the MiniSat solver which won SAT-Race 2006 and has commercial backing from Cadence Design Systems which produces formal verification tools. SAT-decision algorithms have improved tremendously in recent years on problems of interest to industrial applications such as formal verification and MiniSat has been at the forefront of the development [ES04, EB05, EMS07, Een07].

We used the recent 2.0 version for the latest algorithms.

The SAT-races, like much of the SAT literature, are concerned with the decision problem rather than the problem of finding witnesses. Finding witnesses using the decision problem as an oracle requires solving many decision problems. If some of these problems are intractable, it may not be possible to systematically find witnesses. We also note that there is an optimization variant of the SAT problem called MAXSAT [HS00, Hås01, KZ97]. Specifically, it is possible to approximate 3SAT by finding assignments that satisfy most, but not all clauses. Known algorithms (e.g. [KZ97]) are polynomial time for finding an assignment that satisfies 7/8 of the clauses, but become exponential in the worst case when trying to do a full assignment.[1]

One way to find spurious witnesses is to try strings a short Hamming distance from a legitimate witness. The reason is that changing a small number of variables has a better chance of satisfying the instance than a random string. In the experiments described on Section 4.4.2 on page 44, we found such witnesses and computed the likelihood of finding one with a given set of parameters by tallying them for smaller instances and extrapolating. For a Hamming distance of 1 or 2, we have experimentally found $2e^{-3\alpha/7}$ instances with spurious witnesses (see Figure 3 on page 46) given one chosen witness. One might expect this formula to have a factor of ℓ in it. This is not the case for two reasons. One is that if a spurious witness is found with Hamming distance 1, there is a higher probability of spurious witnesses an additional Hamming distance from it. The second reason is that instances which have spurious witnesses typically have many spurious witnesses. These skews to the distribution give us relatively few instances with spurious wit-

[1] For both an algorithm to compute the 7/8-approximation and a proof that MAXSAT can be solved in polynomial time if and only if $P = NP$, see Lecture 18 in [Koz06].

nesses. One might also note that $\alpha \sim \frac{1}{\ell}$. Since we are multiplying α by a negative, that means that the formula increases as the number of variables increase if the number of clauses is kept constant.

Experiments with spurious witnesses at short Hamming distances suggest that the number of witnesses should be further limited to 5. In these experiments, we analyzed more than four million instances with varying parameters to determine what parameters result in few short-Hamming-distance spurious witnesses as described on Section 4.4.2 on page 44. The parameters varied include the clause density and the number of variables. If we relax the security definition to allow an authorized user, rather than an authorized string, to prove membership, short-Hamming-distance attacks cease to be a issue and the system can safely support up to ten witnesses.

It also does not make it easier to solve the instance to put an upper bound on the number of satisfying assignments. The unique satisfiability problem (USAT) is the SAT problem if it is known that there is no more than one satisfying assignment. Finding a solution to an instance of USAT is as hard as SAT [VV85, Koz06]. Furthermore, Valiant and Vazirani show that distinguishing between instances with no solutions or one solution is as hard as SAT.

If there are spurious witnesses, then there are more witnesses than participants. We would like to be able to check if this is the case because if we could, we could rerun the protocol until only the selected witnesses satisfy the formula. Unfortunately, counting the number of satisfying instances for a SAT problem is complete for *#P*, the class of counting problems [CH96]. *#P* (see Section 3.2 on page 19) is a hard class of problems as evidenced by the fact that a Turing machine with a *#P* oracle can decide the polynomial hierarchy in deterministic polynomial time—that is, $PH \subseteq P^{\#P}$ [Tod89, Koz06]. This means that a deterministic Turing machine with

an oracle that can count SAT solutions can decide languages that are Σ_k^P-complete or Π_k^P-complete for any $k \in \mathbb{N}$. Since *NP* is Σ_1^P and co*NP* is Π_1^P it gives a deterministic Turing machine with the oracle the ability to efficiently decide both *NP*-complete and co*NP*-complete languages (see Section 3.2) [Koz06].

There are also combinatorial limits to consider. Although the security analysis of the system dictates a much smaller number of witnesses than the combinatorial maximum, we consider this maximum anyway. The number of possible clauses for a set of witnesses is given by:

$$8\ell(\ell-1)(\ell-2)\left(\frac{7}{8}\right)^m$$

For $\ell = 8,192$ and $n = 65,536$, the combinatorial maximum number of witnesses is 115. We suggest 50 as a practical maximum for the distributed algorithm as the number of elections required by the distributed protocol greatly increases as the system approaches its combinatorial limits. Each election requires computational and communications overhead. When $m = 50$, the system performs nicely, but not securely, as can be seen in the experimental results in Figure 2. The problem with $m = 50$ is that the resulting instances can be easily broken by a SAT solver. For this reason we suggest further restricting the witnesses to $m = 5$. The values for *maxreject* are also shown in Figure 1 and were computed with $S = 2$. This worked with our choice of a seed for the random number generator so that the actual number of rejections never exceeded *maxreject*. Experimental results suggest that the system is only secure for 5 or fewer witnesses. 10 witnesses can be used under an alternate security definition concerned with authorized users rather than witnesses. At 20 witnesses, the system with $\ell = 8,192$ can be solved in less than a second. We could not experimentally solve systems with 10 or fewer witnesses us-

ing the MiniSat solver even with weeks of computing time on a high-performance cluster when $\ell = 8,192$ and $n = 65,536$.

Multiple cheating participants might collude to try to "spread out" their cheating rejections so that they can influence the outcome without exceeding *maxreject*. This can be compensated for by decreasing *maxreject* or by limiting the total number of rejections allowed cumulatively for all participants rather than for individual participants. However, this also increases the chance of "false positives," in which the protocol is restarted even without cheating behavior, so it is only likely to work well for a small number of colluding participants. Having the protocol restarted affects the efficiency of the system, while allowing too many rejections affects the security of the system. It remains open to address other possibilities for cheating and collusions.

4.4.1 Experiments

In all of our experiments, we assumed that the witnesses have a random distribution. If the witnesses do not have a random distribution, security issues may occur depending on the distribution of the witnesses. Our experiments were all carried out on a modern Sun Microsystems cluster consisting of AMD Opteron processors. This state of the art equipment is at Stevens Institute of Technology. Although the experiments were run on a cluster, they were single-threaded and the only parallelism was that of running experiments simultaneously.

In one experiment, we generated instances with 8,192 variables, 524,288 clauses and different numbers of witnesses. We ran MiniSat on these instances to see if they could be decided. These experiments found that with 20 or more witnesses, the instances could be decided very quickly. Furthermore, we found that it appears

to become exponentially easier to decide an instance as the number of witnesses increases. From here, we were able to limit the number of chosen witnesses that we may safely use against MiniSat to 10.

We did further experiments with MiniSat (see Section 4.4.3 on page 48). Although one may be able to decide an instance without learning a witness, our experiments determined if instances with different parameters may be decided efficiently by MiniSat. Since an instance may be tried for many hours without being decided, but may be decided in some slightly longer amount of time, we chose to err on the safe side when specifying the number of witnesses. First we ran experiments with eight instances for four weeks to see if they can be decided in that time. When they weren't decided, we generated one hundred instances with ten witnesses. We had already determined with small experiments that the more witnesses an instance has, the more quickly it can be decided. We took one of the ten witnesses and disallowed it by specifying against it in the instance. This makes it easier for MiniSat to decide an instance. We gave MiniSat fifty hours to try to decide each instance. This was more than adequate given our experience that instances which can be efficiently decided are decided in less than an hour. The purpose of this experiment was to determine the safety of our recommended parameters against the MiniSat program.

None of these were decided. We also did experiments with MiniSat for $\alpha = 128$. For the case of $\alpha = 128$, instances were decided in a few seconds. With $\alpha = 64$, none of the 100 instances tried were decided in the allowed fifty hours for $16,384$ variables.

We also analyzed over four million smaller instances for short-Hamming-distance attacks as described in Section 4.4.2 on the following page. In addition, we analyzed another two hundred thousand instances to verify that the experiments

done on the smaller instances reflect on the larger instances that we recommend in our system.

The purpose of these experiments was to establish three facts. The first was to establish an inverse exponential relationship between the clause density and the probability of an instance having a spurious witness. We expected this probability to be $\Theta(e^{-3\alpha/7})$ for a single chosen witness. This was confirmed by the experiments. The second was to establish how many witnesses the system can safely support. We set an arbitrary level of safety at one instance with a spurious witness in ten thousand, with the intention of erring on the side of caution. In order to render these results valid we had to establish a third fact, which is that if the clause density is held constant, increasing the number of variables will not increase the number of instances with spurious witnesses. This is important as analyzing millions of instances with $16,384$ variables is outside the capability of modern computers. As an additional check we analyzed twelve instances with $16,384$ variables and clause density 64 to Hamming distance 3 in parallel with the other experiments.

4.4.2 Experiments With Short Hamming Distance Attacks

The short-Hamming-distance experiments consisted of three programs. These programs all created instances by first randomly choosing the witnesses and then running the centralized protocol. The first program analyzed instances with a fixed number of variables for all integer values of α from 8 to 128. For each value of α, it tried short-Hamming-distance attacks for $10,000$ random instances and counted how many instances had short-Hamming-distance witnesses. Figure 3 on page 46 contains the number of instances with spurious witnesses with Hamming distance

1 or 2 found for 10,000 runs for each value of α with 1 witness. This involved analyzing more than one million instances for each run. This experiment was done for both 64 and 128 variables bringing the number of instances analyzed to 2 million. The results for 64 and 128 variables were the same with small fluctuations on individual data points reflecting the randomness of the instances. In both cases, the data points corresponded to a probability of an instance having a spurious witness of $2e^{-3\alpha/7}$.

We performed the same experiment, again analyzing more than a million instances, with five witnesses; see Figure 4 on page 47. Once again at $\alpha = 64$ the number of short-Hamming-distance spurious witnesses was negligible. With ten witnesses (see Figure 5 on page 48) there was about one spurious witness for every thousand instances when $\alpha = 64$. We considered this too high and we recommend against using 10 witnesses. These curves had the same shape as the curve for one witness, but we were not able to determine a formula including the number of witnesses. This formula is not necessary as long as we are able to determine the probability of a spurious witness when $\alpha = 64$.

These experiments were done with 64 variables. 64 variables sufficed because we found with the second experiment (below) that the number of variables do not affect the results as long as the clause density remains constant. After finding that ten witnesses were too many, we restricted the system to five and not some intermediate value between five and ten to provide a buffer rather than specifying the system right up to our arbitrary level of safety of one instance with a spurious witness in ten thousand. This cautious approach reflects the arbitrariness of the threshold and the difficulty of measuring the probability that precisely.

The reason that we did the experiments with fewer variables is that the clause density and the number of witnesses are the controlling variables, not ℓ. Our sec-

ond program establishes this. To verify that the above results scale up to large instances, the second program kept α constant and varied the number of variables with 1,000 instances for each value of ℓ from 8 to 128 for a total of more than one hundred thousand instances. This was sufficient for establishing a flat line rather than a curve or a slope. We found that the number of spurious witnesses fluctuated slightly around a flat line as the value of ℓ varied. This suggests that as long as α remains the same, the value of ℓ does not affect the number of spurious witnesses. This was done for $\alpha = 8$ and $\alpha = 16$. Larger values of α generate too few spurious witnesses to read the stability of the data through the noise.

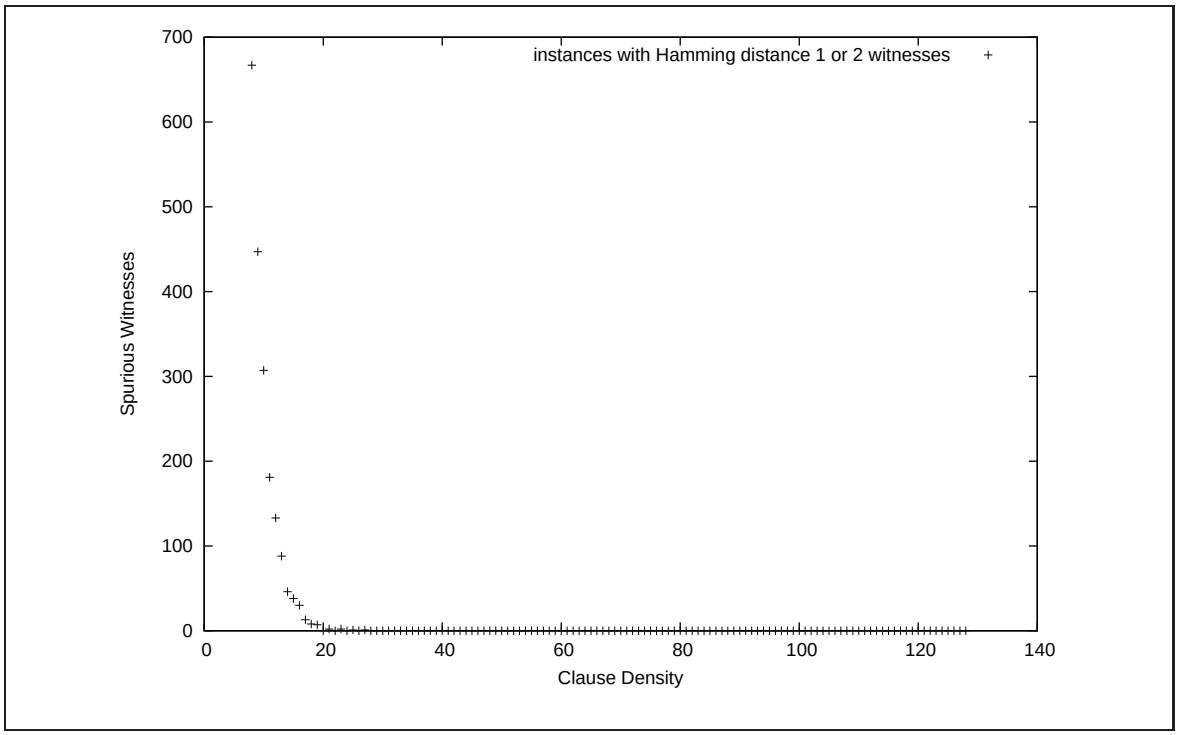

Figure 3: For each α, 10,000 instances with 1 chosen witness were tested for spurious witnesses at Hamming distance 1 or 2.

In parallel, using the third program we tested 12 instances with our recommended parameters for spurious witnesses at Hamming distance 3 or less. None of these large instances had a spurious witness at Hamming distance 3 or less.

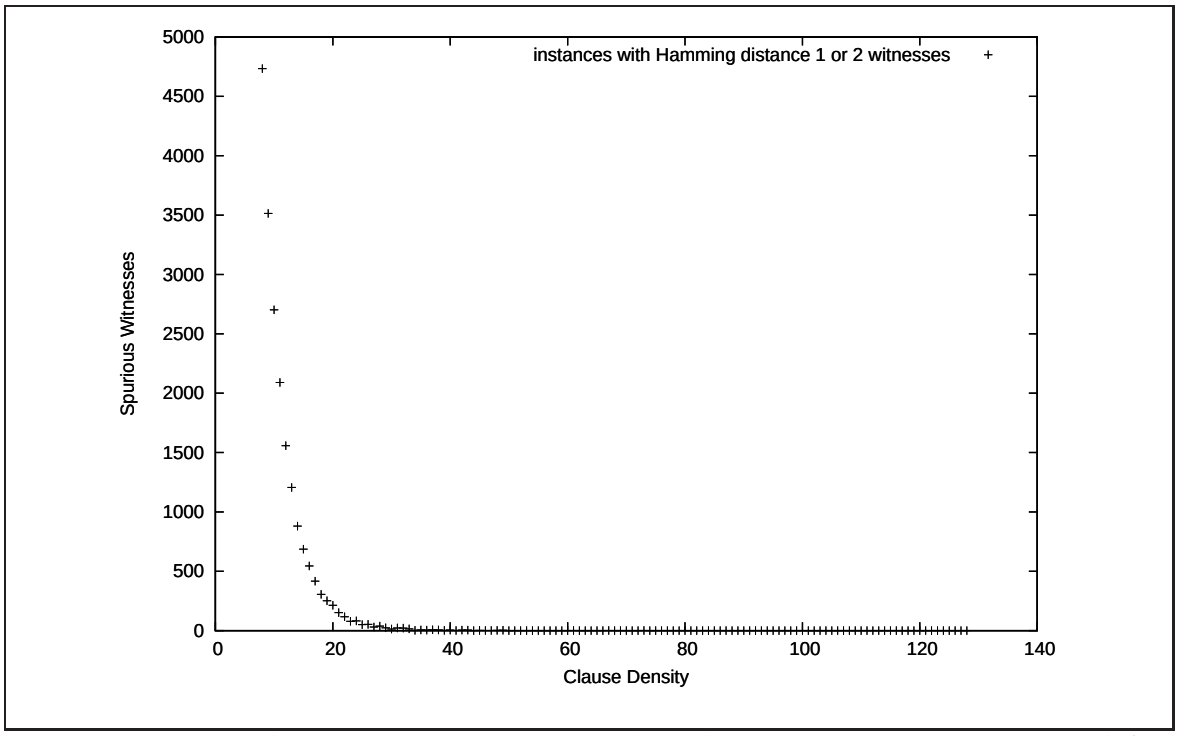

Figure 4: For each α, 10,000 instances with 5 chosen witnesses were tested for spurious witnesses at Hamming distance 1 or 2.

If short-Hamming-distance attacks are not a concern then our recommended parameters appear to be secure for up to 10 witnesses; otherwise, we found that our parameters are secure for up to 5 witnesses. Systems with more than 10 witnesses were found to be insecure. We also experimentally found that increasing α to 128 can result in instances that are easily solved by SAT solvers. This means that we cannot increase the number of witnesses in our system.

One would not be concerned about such spurious witnesses when the provers use a zero-knowledge proof of possession protocol and are trusted not to reveal strings to unauthorized users. Spurious witnesses with a Hamming distance greater than three from any chosen witness are not a concern as they are combinatorially difficult to find as the number of possibilities increase.

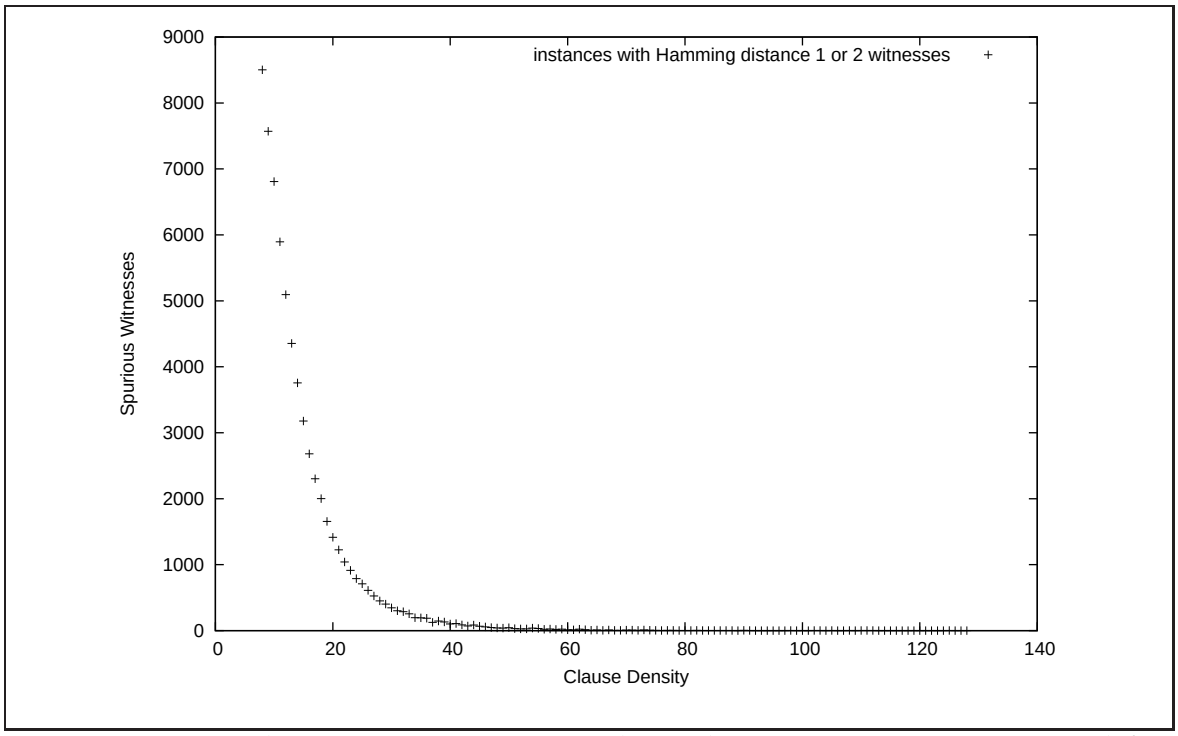

Figure 5: For each α, 10,000 instances with 10 chosen witnesses were tested for spurious witnesses at Hamming distance 1 or 2.

4.4.3 Experiments using MiniSAT

We generated four instances with one witness and four instances with ten witnesses. These instances were generated by first choosing the witnesses randomly and then running the centralized protocol. None of these eight instances were decided, although they ran on the high-performance cluster for more than four weeks. Although it was too few instances to be conclusive, it showed that at least some of the instances generated were hard. To get more conclusive results we ran the following experiment.

We tried 100 instances with ten witnesses with $\ell = 16,384$ and $n = 1,048,576$ also providing one of the witnesses. We provided the witness by appending to the instance to disallow that witness. This experiment was both to determine the safety of the system against finding witnesses in general and to determine the safety of

the system against finding witnesses when a witness is already known.

This experiment determined if knowing one of the witnesses makes it easy enough for MiniSAT to decide the instance by disallowing the known witness. In smaller experiments, we found that this did make it easier for MiniSat to decide an instance. However, with our suggested parameters, MiniSat was unable to determine if there were any other witnesses for any of the instances. In fact, when the experiments were terminated after at least 50 hours of runtime, MiniSat reported that 0.0000% progress had been made. In some cases, the experiment was allowed to run for as much as twice as long. In these cases no progress was made either. This suggests that our system is secure against the MiniSat solver using our suggested parameters and ten or fewer witnesses, but we found that as the number of witnesses increases, it becomes easier for MiniSat to decide the instances, to the point where at 20 witnesses it only takes a second. It is not possible to pinpoint where between ten and twenty it becomes too easy, so we consider ten to be the safe maximum against MiniSat.

Chapter 5

Conclusion

We have presented a general solution to the set membership problem whose security depends on the difficulty of finding witnesses to random 3SAT instances satisfying a given set of witnesses. We have also presented applications to access control, digital credentials, and timestamping. We have shown a distributed protocol for establishing a set.

A strong justification for considering security based on 3SAT is the increased worry that advances in conventional or quantum computing may one day yield efficient algorithms for problems such as factoring and discrete logarithms typically used as a source of hardness in cryptography. It is therefore important to investigate cryptographic algorithms based on alternate (plausible) hardness assumptions to provide resilience against "breaking" of any one assumption or class of assumptions.

In 1979, Brassard showed that if a one-to-one bounded-input one-way function can be shown *NP*-hard to invert, then $NP = \text{co}NP$ [Bra79]. Our system is not one-to-one and does not meet the standard definitions of one-way functions, so Brassard's result does not apply.

Our protocol can be used for digital credentials including anonymous credentials, timestamping, and other set membership applications. It can also be used for applications where multiple users share an account. These include some access control and financial applications. For set membership applications like timestamping, the set representation can be thought of as a distributed signature. It can be proven to any honest participant or observer using the set membership protocol that a document was used for inclusion in the set. These applications have broad applicability to problems in cryptography and security. The advantages of this method over one-way accumulators include not needing to remember a string other than the one chosen by the user and not being dependent on the factoring problem [BdM94].

As previously discussed, the expected number of clauses that must be tried to generate a clause in the set representation is $(\frac{8}{7})^m$, where m is the number of witnesses to be represented. We note that this probability depends on the number of witnesses and is independent of n and ℓ. In contrast, the security of the system is based on the adversary's difficulty of finding an element as a function of n and ℓ, so it may be possible to limit m so as to have efficient solutions for the participants without making the adversary's task solvable. As described earlier in Section 4.1, we believe that one hundred witnesses can be dealt with easily, but that as the number of witnesses begins to reach one hundred fifty, it becomes infeasible to generate an instance. It is important to remember that for our recommended parameters, the system is not secure with more than five witnesses.

The space complexity for a set based on 3SAT is $\Theta(\ell \log \ell)$. For instance, a system with $8,192$ variables requires $65,536$ clauses. Altogether, this requires 128 kilobytes of storage. For our recommended parameters, 256 kilobytes of storage are required. This space complexity is independent of the number of set elements.

The secure set membership primitive solves the secure set membership problem. Secure set membership is a generalization of the problem solved by one-way accumulators [BdM94]. Our secure set membership primitive uses a computational problem based on 3SAT that is not known to be efficiently computable by either classical or quantum computers. We show how to generate this primitive using a distributed protocol and how to prove possession of a set member without revealing the string.

The secure set membership primitive is weaker than one-way functions, and cannot be used to build one-way functions or more complicated primitives. Still, it is our belief that by starting with a simple, weak primitive that more useful new primitives will be developed, possibly using our complexity assumption.

Bibliography

[ABS03] Mikhail Alekhnovich and Eli Ben-Sasson. Linear upper bounds for random walk on small density random 3-CNFs. In *Proceedings of the 44th Annual IEEE Symposium on the Foundations of Computer Science*, 2003.

[Acq03] Alessandro Acquisiti. Anonymous credentials through acid mixing. Unpublished manuscript, 2003.

[AGGM06] Adi Akavia, Oded Goldreich, Shafi Goldwasser, and Dana Moshkovitz. On basing one-way functions on NP-hardness. In *Proceedings of the 38th Annual ACM Symposium on Theory of Computing (STOC '06)*. ACM Press, 2006.

[AHU74] Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974.

[Ajt04] Miklos Ajtai. Generating hard instances of lattice problems. *Complexity of Computations and Proofs, Quaderni di Matematica*, 13(1-32), 2004.

[BC87] Gilles Brassard and Claude Crepeau. Zero-knowledge simulation of boolean circuits. In *Advances in Cryptography – Crypto '86*, volume

263/1987 of *Lecture Notes in Computer Science*, pages 223–233. Springer, 1987.

[BdM91] Josh Benaloh and Michael de Mare. Efficient broadcast time-stamping. Technical Report TR-MCS-91-1, Clarkson University Department of Mathematics and Computer Science, 1991.

[BdM94] Josh Benaloh and Michael de Mare. One-way accumulators: A decentralized approach to digital signatures. In *Advances In Cryptology – Eurocrypt '93*, volume 765/1994 of *Lecture Notes in Computer Science*, pages 274–285. Springer, 1994.

[Ben87] Josh Benaloh. *Verifiable Secret-Ballot Elections*. PhD thesis, Yale University Department of Computer Science, September 1987.

[BF03] Dan Boneh and Matthew Franklin. Identity-based encryption from the Weil pairing. *SIAM Journal of Computing*, 32(3):586–615, 2003.

[BP97] Niko Baric and Birgit Pfitzmann. Collision-free accumulators and fail-stop signature schemes without trees. In *Advances in Cryptology – Eurocrypt '97*, volume 1233/1997 of *Lecture Notes in Computer Science*. Springer, 1997.

[Bra79] Gilles Brassard. A note on the complexity of cryptography. *IEEE Transactions on Information Theory*, IT-25(2):232–233, March 1979.

[BT03] Andrej Bogdanov and Luca Trevisan. On worst-case to average-case reductions for np problems. In *44th IEEE Conference on the Foundation of Computer Science (FOCS 2003)*, pages 308–317. IEEE, 2003.

[CH96] Nadia Creignou and Miki Hermann. Complexity of generalized satisfiability counting problems. *Information and Computation*, 125(1):1–12, February 1996.

[CL01] Jan Camenisch and Anna Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In *Advances in Cryptology – Eurocrypt 2001*, volume 2045/2001 of *Lecture Notes in Computer Science*. Springer, 2001.

[CL02] Jan Camenisch and Anna Lysyanskaya. Dynamic accumulators and application to efficient revocation of anonymous credentials. In *Advances in Cryptology – Crypto 2002*, volume 2442/2002 of *Lecture Notes in Computer Science*. Springer, 2002.

[CLRS01] Thomas Corman, Charles Leiserson, Ronald Rivest, and Clifford Stein. *Introduction to Algorithms*. Prentice-Hall, 2nd edition, 2001.

[DH76] Whitfield Diffie and Martin Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, Nov. 1976.

[dM04] Michael de Mare. An analysis of certain cryptosystems and related mathematics. Master's thesis, State University of New York Institute of Technology, Dec. 2004.

[dM08] Michael de Mare. Iterative symmetric-key ciphers with keyed S-boxes using modular exponentiation. Patent application 12/051,626, 2008.

[dMW06] Michael de Mare and Rebecca N. Wright. Secure set membership using 3SAT. In *Proceedings of the Eighth International Conference in Infor-*

mation and Communications Security (ICICS 2006), volume 4307/2006 of *Lecture Notes in Computer Science*, pages 452–468. Springer, 2006.

[DSW94] Martin D. Davis, Ron Sigal, and Elaine J. Weyuker. *Computability, Complexity, and Languages: Fundamentals of Theoretical Computer Science*. Computer Science and Scientific Computing. Morgan Kaufmann, second edition, 1994.

[EAFH04] Fernando Esponda, Elena S. Ackley, Stephanie Forrest, and Paul Helman. On-line negative databases. In *Proceedings of the 3rd International Conference on Artificial Immune Systems (ICARIS)*, volume 3239/2004 of *Lecture Notes in Computer Science*, pages 175–188. Springer, Sep. 2004.

[EB05] Niklas Een and Armin Biere. Effective preprocessing in SAT through variable and clause elimination. In *Theory and Applications of Satisfiability Testing 8th International Conference (SAT 2005)*, volume 3569/2005 of *Lecture Notes in Computer Science*, pages 61–75. Springer, 2005.

[Een07] Niklas Een. Cut sweeping. Technical report, Cadence Design Systems, 2007.

[EFH04a] Fernando Esponda, Stephanie Forrest, and Paul Helman. Enhancing privacy through negative representations of data. Technical report, University of New Mexico, 2004.

[EFH04b] Fernando Esponda, Stephanie Forrest, and Paul Helman. Information hiding through negative representations of data. Technical report, University of New Mexico, 2004.

[EMS07] Niklas Een, Alan Mishchenko, and Niklas Sörensson. Applying logic synthesis for speeding up SAT. In *Theory and Applications of Satisfiability Testing – SAT 2007*, volume 4501/2007 of *Lecture Notes in Computer Science*, pages 272–286. Springer, 2007.

[ES04] Niklas Een and Niklas Sörensson. An extensible SAT-solver. In *Sixth International Conference on Theory and Applications of Satisfiability Testing (SAT 2003)*, volume 2919/2004 of *Lecture Notes in Computer Science*, pages 502–518. Springer, 2004.

[Esp05] Fernando Esponda. *Negative Representations of Information*. PhD thesis, University of New Mexico, 2005.

[FF93] Joan Feigenbaum and Lance Fortnow. Random self-reducibility of complete sets. *Siam Journal on Computing*, 22:994–1005, 1993.

[GJ79] Michael Garey and David Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, 1979.

[GMMU07] Robert Gilman, Alexei G. Miasnikov, Alex D. Myasnikov, and Alexander Ushakov. Report on generic case complexity. Technical report, Stevens Institute of Technology, March 2007. Algebraic Cryptography Center.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to prove all NP statements in zero-knowledge and a methodology of cryptographic protocol design. In *Advances in Cryptology – Crypto '86*, volume 263/1987 of *Lecture Notes in Computer Science*, pages 171–185. Springer, 1987.

[Hås01] Johan Håstad. Some optimal inapproximability results. *J. ACM*, 48(4):798–859, 2001.

[HMU01] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Pearson Addison Wesley, 2001.

[HS00] Holger H. Hoos and Thomas Stützle. SATLIB: An online resource for research on SAT. In *Third Workshop on the Satisfiability Problem (SAT 2000)*, pages 283–292. IOS Press, 2000. see also `http://www.satlib.org`.

[IM03] Russell Impagliazzo and Sara Miner. Anonymous credentials with biometrically-enforced non-transferability. In *Proceedings of the 2003 ACM Workshop on Privacy in the Electronic Society*, pages 60–71. ACM, 2003.

[IR89] Russell Impagliazzo and Steven Rudich. Limits on the provable consequences of one-way permutations. In *Proceedings of 21st Annual ACM Symposium on the Theory of Computing*, pages 44–61, 1989.

[Knu98] Donald E. Knuth. *Sorting and Searching*, volume 3 of *The Art of Computer Programming*. Addison-Wesley, second edition, 1998.

[Koz92] Dexter Kozen. *The Design and Analysis of Algorithms*. Texts and Monographs in Computer Science. Springer, 1992.

[Koz06] Dexter Kozen. *Theory of Computation*. Texts in Computer Science. Springer, 2006.

[KZ97] Howard J. Karloff and Uri Zwick. A 7/8-approximation algorithm for MAX 3SAT. In *Proceedings of the 38th Annual Symposium on Foundations of Computer Science (FOCS '97)*, pages 406–415. IEEE Computer Society, 1997.

[LdW05] Arjen Lenstra and Benne de Weger. On the possibility of constructing meaningful hash collisions for public keys. In *The 10th Australasian Conference on Information Security and Privacy: ACISP 2005*, volume 3574/2005 of *Lecture Notes in Computer Science*, pages 267–279. Springer, 2005.

[Lyn96] Nancy A. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers, 1996.

[Mer82] Ralph C. Merkle. *Secrecy, authentication, and public key systems*. UMI Research Press, 1982.

[Mer88] Ralph C. Merkle. A digital signature based on a conventional encryption function. In *Advances in Cryptology – Crypto '87*, volume 293/1988 of *Lecture Notes in Computer Science*. Springer, 1988.

[Mic07] Daniele Micciancio. Cryptographic functions from worst-case complexity assumptions. In *LLL+25*, 2007.

[MvOV97] Alfred Menezes, Paul C. van Oorschot, and Scott Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1997.

[Odl90] Andrew M. Odlyzko. The rise and fall of the knapsack cryptosystems. In *PSAM: Proceedings of the 42nd Symposium in Applied Mathematics*, pages 75–88, 1990.

[Pap95] Christos H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1995.

[Rab83] Michael Rabin. Transaction protection by beacons. *Journal Computer and System Sciences*, 27(2):256–267, October 1983.

[RSA78] Ronald Rivest, Adi Shamir, and Leonard Adleman. A method for obtaining digital signatures and public key cryptosystems. *Communications of the ACM*, 21(2):120–126, Feb. 1978.

[Rud88] Steven Rudich. *Limits on the Provable Consequences of One-way Functions*. PhD thesis, EECS Department, University of California, Berkeley, 1988.

[Sho94] Peter Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings of the 35th Annual Symposium on Foundations of Computer Science*, pages 124–134. IEEE, 1994.

[Sin06] Carston Sinz. SAT-Race 2006, 2006. International Conference on Theory and Applications of Satisfiability Testing (SAT06), See also: `http://fmv.jku.at/sat-race-2006/`.

[Szy04] Michael Szydlo. Merkle tree traversal in log space and time. In *Advances in Cryptology – Eurocrypt 2004*, volume 3027/2004 of *Lecture Notes in Computer Science*, pages 541–554. Springer, 2004.

[Tod89] Seinosuke Toda. On the computational power of *PP* and $\oplus P$. In *Proceedings of the 30th Symposium of the Foundations of Computer Science (FOCS '89)*, pages 514–519. IEEE, 1989.

[VV85] Leslie G. Valiant and Vijay V. Vazirani. *NP* is as easy as detecting unique solutions. In *Proceedings of the 17th ACM Symposium on Theory of Computing (STOC '85)*, pages 458–463. ACM, 1985.

[Wat06] John Watrous. Zero knowledge against quantum attacks. In *Proceedings of the 38th Annual ACM Symposium on Theory of Computing (STOC '06)*, pages 296–315. ACM Press, 2006.

[WYY05a] Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu. Finding collisions in the full SHA-1. In *Advances in Cryptography – Crypto 2005*, volume 3621/2005 of *Lecture Notes in Computer Science*, pages 17–36. Springer, 2005.

[WYY05b] Xiaoyun Wang, Hongbo Yu, and Yiqun Lisa Yin. Efficient collision search attacks on SHA-0. In *Advances in Cryptography – Crypto 2005*, volume 3621/2005 of *Lecture Notes in Computer Science*, pages 1–16. Springer, 2005.

Index

Vita

Michael de Mare was born in Brooklyn, New York on October 14, 1969. He has a Bachelor of Science from Clarkson University in Computer Science awarded May 1991, a Master of Science from State University of New York Institute of Technology in Computer Science awarded December 2004, and is a doctoral candidate at Stevens Institute of Technology in Computer Science. He is currently a Visiting Instructor of Computer Science at State University of New York Institute of Technology. He worked as a systems analyst at Giordano Automation from 1992 to 1994, as a senior software engineer at Ikos Systems from 1994 through 1995, for Technoproductions Inc. (consulting for Ikos Systems) from 1996 through 1998, and as a senior software engineer at LSI Logic from 1998 through 2000. His publications include *Efficient Broadcast Timestamping*, a technical report at Clarkson University, [BdM91], *One-Way Accumulators: A Decentralized Approach to Digital Signatures* at Eurocrypt '93, [BdM94] and *Secure Set Membership Using* 3SAT at the Eighth International Conference on Information and Communications Security (ICICS 2006), [dMW06], as well as his masters thesis: *An Analysis of Certain Cryptosystems and Related Mathematics*, [dM04] and the patent application: *Iterative Symmetric-Key Ciphers With Keyed S-boxes Using Modular Exponentiation*, [dM08]. This patent covers the Dragonfire cipher, polymorphic S-boxes, and pseudoindependent subkeys. He was a National Merit Scholar Letter of Commendation winner and won an Engi-

neering Excellence Award at LSI Logic.