Efficient Commitment Schemes with Bounded Sender and Unbounded Receiver

Shai Halevi

MIT – Laboratory for Computer Science,
545 Technology Square, Cambridge, MA 02139
shaih@theory.lcs.mit.edu

Abstract. In this paper we address the problem of commitment schemes where the sender is bounded to polynomial time and the receiver may be all powerful. We present a scheme for committing to a (possibly long) string. Our scheme is efficient in the following three ways:

ROUND EFFICIENCY: Each part of the scheme consists of a single round.

LOW COMMUNICATION: The number of bits required for the commitment equals the security parameter of the system, regardless of the length of the string which is being committed to.

FAST IMPLEMENTATION: The time taken to commit to a string is linear in the length of the string and almost linear in the security parameter of the system.

1 Introduction

In this paper we address the problem of commitment schemes for (possibly long) messages. The problem arises when Alice has a message which Bob does not know, and they want to simulate (by means of electronic communication) the effect of delivering the message to Bob in a sealed envelope (or better yet, in a locked box): Alice wants to prevent Bob from knowing anything about the message in the box until such time in the future when she decides to give him the key. Bob, on the other hand, wants to prevent Alice from changing the message in the box after he has already received it.

COMMITMENT SCHEMES. A protocol for implementing such a simulation is called a commitment scheme. It consists of two phases. The first phase simulates the delivery of the locked box. When this phase is completed, Bob does not know the message yet, but Alice can not change it any more. The second phase simulates the delivery of the key. Bob can now see the message and verify that it is indeed the message to which Alice is committed.

EXAMPLE. As a simple example of a commitment scheme, consider a public-key encryption function $E(\cdot)$. To commit to a message σ, Alice sends the encryption $c = E(\sigma)$ to Bob. After this is done, Bob still does not know what σ is (provided that he can not break the encryption) but Alice can not change it anymore since there is no other message which encrypts to c. When Alice wants to reveal

D. Coppersmith (Ed.): Advances in Cryptology - CRYPTO '95, LNCS 963, pp. 84-96, 1995.

her message, she just send it to Bob, who encrypts it and checks that it really encrypts to c.

COMMITMENT SCHEMES WITH COMPUTATIONALLY UNBOUNDED BOB. The above scheme can work only if Bob is computationally bounded, so he can not break the encryption. If Bob has unbounded computational power, then Alice needs to commit in a way that yields no information (in the information theoretic sense) about her message. Of course, in this case there are many different messages that correspond to the same commitment. Thus, Alice must be computationally bounded, so she can not find any of the other messages. This is the case that we address in this paper.

RUNNING-TIME. Although it is known that commitment schemes exists based on "weak" assumptions, the implementations of such schemes are still based on either the hardness of factoring or the hardness of discrete log. In the discrete-log based implementations, a typical operation requires a modular exponentiation which is a relatively expensive operation. Thus these implementations are usually less efficient than the factoring-based ones, where a typical operation requires only modular multiplication. The scheme that we present in this paper is of the latter kind.

1.1 Previous Work

Commitment schemes were first formulated by Blum in the context of flipping coins over the telephone. The problem which is considered in this context is committing to a single bit. The first implementation of a bit-commitment scheme for unbounded Alice and bounded Bob (which was based on the hardness of discrete log) was suggested in [BM81] (as cited in [Blu82]). A different technique based on the hardness of factoring was embedded in [GM84].

The first bit-commitment scheme for the case where Bob is computationally unbounded was described by Blum in [Blu82]. This scheme is based on the hardness of factoring. In the same paper Blum also introduced the use of "Blum integers" (i.e., product of two primes, both congruent to 3 mod 4), which were used in many other papers since, including this one.

Since then there has been a large body of research regarding the bit commitment problem. In particular, it was shown that such bit commitment schemes exist in various models, based on various assumptions. For example, see [Nao90, Dam90, BC91, DPP94, IOS94].

In [Nao90], Naor also considered the problem of committing to long messages in the case of unbounded Alice and bounded Bob. Based on the existence of pseudo random generators, he describes a very elegant commitment scheme for long strings which only uses $O(n)$ bits to commit to a string of length n (where the constant in the $O(\cdot)$ does not depend on the security parameter of the system).

In addition, there have been much work on using various bit-commitment schemes within cryptographic protocols and zero-knowledge proofs. For example, see [BM84, GMW91, BCC88, BMO90, NOVY92].

1.2 Contributions of This Paper

In this paper we address the problem of committing to (possibly long) messages where Alice is bounded and Bob is unbounded. Although it is possible to use bit-commitment schemes to commit to a longer message by committing to each bit separately, the performance of such schemes is quite bad: The protocols for bit commitment require k bits of commitment for every message bit (in a system with security parameter k). If the message is long, then sending and storing such a large commitment may be a problem.

COMMUNICATION, ROUNDS AND RUNNING-TIME EFFICIENCY. We present a scheme where the length of the commitment string *does not depend on the length of the message*. The number of bits it takes Alice to commit to any string equals the security parameter of the system, regardless of the length of that string. Our scheme is also efficient in terms of round complexity. Each part of the scheme consists of a single round.

The scheme we present in this paper uses the Goldwasser-Micali-Rivest claw-free permutation pairs ([GMR88]) which are based on the hardness of factoring. As was mentioned above, each operation in the scheme consists of a modular multiplication, which can be performed in time almost linear in the size of the numbers involved. The scheme requires one or two multiplications for every character in the message.

SIMPLE INITIALIZATION. Our scheme has an advantage over other factoring-based schemes even when committing to just one bit. In many of the known factoring-based schemes, the composite numbers which are used in the scheme must be "Blum integers" (i.e., they have to be products of two primes, both congruent to 3 mod 4). If the numbers are not of the right form, then the security of both parties may be compromised.

Therefore these schemes require additional tools to ensure that the numbers are of the right form (such as using zero-knowledge proofs). These tools are typically very expensive, so the schemes become less efficient.

We present a new technique which eliminates the need for such expensive initialization steps. Our scheme is unique in that the use of "Blum integers" effects only the security of Bob. Thus, we can simply let Bob choose the number and send it to Alice, knowing that the security of Alice does not depend on which number was chosen.

1.3 Organization of the Paper

The rest of this paper is organized as follows: In Sect. 2 we define the notion of a commitment scheme. In Sect. 3 we present our factoring-based scheme which uses the claw-free permutation pairs due to [GMR88]. We first present a very simple implementation and then show how it can be modified to allow simple initialization.

In Sect. 4 we show how we can generalize our scheme and implement it using any construction of claw-free families of permutations.

2 Commitment Schemes

THE SYNTACTIC STRUCTURE OF A COMMITMENT SCHEME. A commitment scheme is a two phase protocol between two parties, Alice and Bob. Both parties share a common input, 1^k (for some integer k) which indicates the security parameter of the system. Besides 1^k, Alice also has another input, σ, which is the message string to which she wants to commit herself. When used inside some other protocol, the parties may also have other inputs which represent their history at the point where the commitment scheme in being invoked.

The scheme itself consists of two phases: The *commit* phase and the *reveal* phase. The parties execute the commit phase first and the reveal phase at some later time. Typically, when used in another protocol, there will be some other parts of that protocol between the commit and the reveal phases.

During the commit phase Alice sends to Bob a commitment string c and during the reveal phase Alice sends to Bob a reveal string r. From c and r Bob computes the message σ and then checks that σ is consistent with c and r.

In the construction which we present below we need an "initialization phase" before we can use the scheme. This phase is independent of the message σ which Alice wants to commit to. In fact, we can execute the initialization phase only once and then use the system to commit to many different messages. Alternatively, we can add the initialization to the commit phase and execute it as part of the protocol. In the implementations that we discuss in this paper, the initialization can be executed quite efficiently.

THE SEMANTICS OF A COMMITMENT SCHEME. Intuitively, the commit phase has the effect of sending the message from Alice to Bob in a locked box. Bob does not yet know anything about the contents of the message, but Alice can not alter the message anymore. The reveal phase has the effect of giving Bob the key and revealing the message inside the box.

The definition of what it means for Bob "not to know anything about σ", and for Alice "not to be able to alter σ" depends on the computational power of the parties. In the context of this paper, Alice is bounded to probabilistic polynomial-time and Bob has unbounded computational power. Thus, we require the following properties

Meaningfulness: If both Alice and Bob follow their parts in the protocol, then the message σ which Bob computes from (c, r) after the reveal phase is equal to Alice's input message.

Security: The communication between Alice and Bob in the commit phase gives no information (in the information-theoretic sense) about σ.

Non-Ambiguity: It is computationally infeasible for Alice to generate a commitment string c and two reveal strings r, r' such that in the reveal phase, Bob would compute one message σ from (c, r), a different message $\sigma' \neq \sigma$ from (c, r') and would accept both (σ, c, r) and (σ', c, r').
This means that for any probabilistic polynomial-time algorithm, the probability of generating c, r, r' as above when given the input 1^k (the security parameter) is negligible.

3 A Factoring-Based Implementation

In this section we present a specific implementation which uses the claw-free permutation families due to [GMR88].

3.1 The Goldwasser-Micali-Rivest Claw-Free Permutation Pairs

Let p and q be two primes such that $p = 3 \pmod 8$, $q = 7 \pmod 8$, and denote $N \stackrel{\text{def}}{=} p \cdot q$. We start by defining two functions

$$f_{N,0}(x) \stackrel{\text{def}}{=} x^2 \pmod N \quad \text{and} \quad f_{N,1}(x) \stackrel{\text{def}}{=} 4x^2 \pmod N$$

Then, for any string $s = b_1 b_2 \cdots b_n$ we define $f_{N,s}(x) \stackrel{\text{def}}{=} f_{N,b_1}(\cdots f_{N,b_n}(x) \cdots)$. It is easy to see that both $f_{N,0}$ and $f_{N,1}$ are permutations over the squares mod N, which implies that for any s the function $f_{N,s}$ is also a permutations over the squares mod N.

3.2 Using the GMR Construction for Commitment

The following is a simple commitment scheme that uses the GMR construction. We assume that Alice and Bob uses some standard encoding function Enc, with the property that for no two messages $\sigma \neq \sigma'$ is $Enc(\sigma)$ a prefix of $Enc(\sigma')$.

Initialization: Alice and Bob "choose at random" a composite N with k bits of the above form. We discuss this phase in more details below.

Commit phase: Given a message σ, Alice computes $s = Enc(\sigma)$. Then she picks a random element $x \in Z_N^*$ and sends $y = f_{N,s}(x^2)$ to Bob.

Reveal Phase: Alice sends both σ and x to Bob. Bob computes $s = Enc(\sigma)$ and verifies that $y = f_{N,s}(x^2)$.

To show that this is a commitment scheme we need to show two things:

Claim 1. *The value of y does not give any information about σ.*

Proof. (sketch) Since both $f_{N,1}$ and $f_{N,0}$ are permutations, then so is $f_{N,s}$ for any s. Thus, for every y (which is a square mod N) and every s there exists exactly one square mod N x such that $y = f_{N,s}(x)$. This implies the claim. □

Claim 2. *If it is infeasible to factor composite numbers of the above form, then it is infeasible for Alice to generate on input N two strings s, s' (none of which is a prefix of the other) and $x, x' \in Z_N^*$ such that $f_{N,s}(x^2) = f_{N,s'}(x'^2)$.*

This claim was proven in [GMR88] (Theorem 1) and a generalization of it was proven in [Dam88] (Theorem 2.8). □

3.3 Efficiency of the Scheme

The amount of communication in the commit phase is independent of σ. Alice always send exactly k bits to Bob (where k is the number of bits in N). In the reveal phase, Alice sends the message σ and k more bits.

In terms of running time, to compute the commitment string Alice needs to perform one or two modular multiplications for every bit in s (which presumably has about the same length as σ). Using construction similar to [Dam88], we can use larger families of permutations to reduce the number of multiplication to one or two per byte (or even word) of s. However, we pay for this by having to keep many more bits to describe these larger families of permutations, and by having to choose one of these families in the initialization phase.

3.4 Implementing the Initialization Phase

The main problem with the above scheme is the implementation of the initialization phase. Clearly, it is important to choose the composite number N in such a way that Alice will not be able to factor it easily. Notice that it doesn't matter whether Bob knows the factorization of N or not.

One idea is to let Bob choose N in the appropriate way and send it to Alice. But if Alice doesn't know the factorization of N, how can she verify that N it is really a product of two primes which are 3 mod 4 (which is the property that makes the functions $f_{N,0}, f_{N,1}$ permutations) ?

At first glance this may not look like a real problem. After all, Alice can choose the starting point x at random, so she may be able to hide σ from Bob even if these functions are not permutations. Unfortunately, this is not the case.

Consider for example $N = 5$ and a message of one bit b. It is easy to see that for any element $x \in Z_5^*$ we have $f_{5,0}(x^2) = 1$ and $f_{5,1}(x^2) = 4$. Thus Bob can recover the message from the commitment string.

To solve this problem Bob can choose N and then prove (by means of a zero-knowledge proof) to Alice that it is of the right form. However this zero-knowledge proof can be expensive in terms of both running time and communication. Moreover, some zero-knowledge proofs use commitment schemes as basic primitives.

It will therefore be desirable to have a system where choosing a "bad N" does not help Bob getting any information about σ. We present such a system below.

3.5 A Modification of the GMR-Based Scheme

The only difference between the following scheme and previous one is that after computing $y = f_{N,s}(x^2)$, Alice squares y k more times (where k is the number of bits in N) and sends the result to Bob. The new scheme is:

Initialization: Bob picks at random an odd k-bit composite number N which is a product of two large primes, one congruent to 3 mod 8 and the other congruent to 7 mod 8. Bob sends N to Alice, who just verifies that N is odd.

Commit phase: Given a message σ, Alice computes $s = Enc(\sigma)$. Then she picks a random element $x \in Z_N^*$ and sends $y = f_{N,0^k s}(x^2)$ to Bob.

Reveal Phase: Alice sends both σ and x to Bob. Bob computes $s = Enc(\sigma)$ and verifies that $y = f_{N,0^k s}(x^2)$.

It is easy to see that if Bob picks N according to the protocol then it is still infeasible for Alice to find two different messages with the same commitment string (if factoring is hard).

The hard part is to show that even if Bob tries to "cheat" by picking a "bad" N, he still does not get any information about σ from the commitment string.

3.6 Proof of Security for the Modified Scheme

Let N be an odd integer and denote the number of bits in N by k. We model Bob's view of the protocol as an experiment in which Alice picks a string $\sigma \in \{0,1\}^*$ according to some distribution D (D represents the knowledge that Bob has about σ). Then Alice computes $s = Enc(\sigma)$, picks at random an element $x \in Z_N^*$ and sends $y = f_{N,0^k s}(x^2)$ to Bob.

Denote by S, X the random variables which take on the values of s, x respectively in the experiment above. The following lemma asserts that y does not give any information about s if we do not know x.

Lemma 3. *For any element $y \in Z_n^*$ such that $\Pr[f_{N,0^k S}(X^2) = y] > 0$ and for any string s we have*

$$\Pr_{S,X}\left[S = s \mid f_{N,0^k S}(X^2) = y\right] = \Pr_{S}[S = s]$$

See Appendix A for a detailed proof. The idea is that all the information which $y = f_{N,s}(x^2)$ gives about s depends only on a property that we call the "tag of y mod N". Moreover, by repeated squaring we force the tag of $y = f_{N,0^k s}(x^2)$ to be some constant which does not depend on s or x. Thus, y does not give any information about s.

Unfortunately, the formal proof is somewhat lengthy. On the up side, it contains some number-theoretic lemmas which may be interesting in their own right.

4 Using General Claw-Free Permutation Families

In this section we show how the above scheme can be generalized to use any construction for claw-free permutation families.

4.1 Claw-Free Families of Permutations

The notion of claw-free permutations families which we use here is a little more general than the one in [GMR88] but still not as general as in [Dam88]. A construction of claw-free permutation families consists of the following components:

1. A constant r which indicates the number of permutations in each family.
2. A set $INDEX_k$ for all $k \in \mathcal{N}$. Every string in $INDEX_k$ is an index of a family with security parameter k. $INDEX_k$ is polynomial-time samplable given 1^k.
3. For every k and every index $\tau \in INDEX_k$ there is a domain Dom_τ that is associated with τ. It is convenient to assume that if $\tau \in INDEX_k$ then $Dom_\tau \subseteq \{0,1\}^k$. Dom_τ is polynomial-time samplable given τ.
4. For every k and every index $\tau \in INDEX_k$ we have a family of functions $\mathcal{F}_\tau = \{f_{(\tau,0)}, f_{(\tau,1)}, \cdots, f_{(\tau,r-1)}\}$ such that all the $f_{(\tau,i)}$'s are permutations over Dom_τ, and there is an efficient algorithm $COMPUTE(\tau, i, x)$ which computes $f_{\tau,i}(x)$ given τ, i and an element $x \in Dom_\tau$.
5. What makes these families claw-free is that the following task is infeasible: Given 1^k and a random element $\tau \in INDEX_k$, find $i \neq j$ and two elements $x, y \in Dom_\tau$ such that $f_{\tau,i}(x) = f_{\tau,j}(y)$.

Notice that the set "legal indexes" in the above definition may or may not be polynomial-time recognizable (i.e. $INDEX = \bigcup_k INDEX_k$ may or may not be in BPP). For example, for the GMR construction it is not known whether $INDEX \in BPP$. On the other hand, it is easy to come up with a simple construction based on the hardness of discrete-log for which $INDEX \in BPP$. As it turns out, if we have a construction for which $INDEX \in BPP$, then the initialization phase for the scheme which we present below becomes much simpler.

4.2 Commitment-Schemes and Claw-Free Permutation Families

Assume that we have a construction of claw-free families of permutations. For any index τ and string $s = b_1 b_2 \cdots b_n$ we denote by $f_{\tau,s}$ the function $f_{\tau,s}(x) \stackrel{\text{def}}{=} f_{\tau,b_1}(\cdots f_{\tau,b_n}(x)\cdots)$. Here is how we use the claw-free families to implement a commitment scheme. On common input 1^k:

Initialization: Alice and Bob pick a family of permutations with security parameter k (by choosing a random index $\tau \in INDEX_k$). We discuss ways to implement this phase below.

Commit phase: Given a message σ, Alice computes $s = Enc(\sigma)$. Then she picks a random element $x \in Dom_\tau$ and sends $y = f_{\tau,s}(x)$ to Bob.

Reveal Phase: Alice sends both σ and x to Bob. Bob computes $s = Enc(\sigma)$ and verifies that $y = f_{\tau,s}(x)$.

Notice that if we have families with more than two permutations, we can use techniques similar to those in [Dam88] to save time by viewing s as a string over an alphabet with more than two symbols: For example, if we have 256 permutations in each family we can view s as a sequence of bytes. This way we only need to apply $f_{(\tau,\cdot)}$ once for every byte in s rather than once for every bit.

The proof that this is indeed a commitment scheme is similar to the proofs of Claims 1 and 2.

4.3 Implementing the Initialization Phase

We consider two different cases here:

Case 1: $INDEX \in BPP$ for this construction. In this case Bob can simply choose a random index $\tau \in INDEX_k$ and send it to Alice. Alice can verify that τ is indeed an index of some permutation family.

Notice that Alice doesn't care how τ was chosen. The fact that all the functions $f_{\tau,i}(\cdot)$ are permutations over Dom_τ is enough to ensure that Bob does not get any information about s from the commitment string. On the other hand, it is in the best interest of Bob to pick τ at random, since the infeasibility condition only applies when τ is chosen at random.

Case 2: $INDEX \notin BPP$ for this construction. In this case Alice can not verify that the functions $f_{\tau,i}(\cdot)$ are permutations, so Bob may choose τ so as to be able to extract information about s from $f_{\tau,s}(x)$.

In this case the parties either need to rely on a trusted party that will pick τ for them, or Bob can prove to Alice that τ is indeed in $INDEX_k$. Another possibility is to modify the scheme itself (as we did in the GMR-based implementation) to eliminate this problem.

5 Acknowledgments

I thank Silvio Micali for many helpful ideas and discussions and Shafi Goldwasser for several very helpful comments. I also thank the MIT dental-service for making me wait for an hour on the dentist chair, in which time I came up with the idea for this paper.

References

[BC91] G. Brassard and C. Crèpeau. Quantum bit commitment and coin tossing protocols. In A.J. Menezes and S. A. Vanstone, editors, *Proceedings CRYPTO 90*, pages 49–61. Springer-Verlag, 1991. Lecture Notes in Computer Science No. 537.

[BCC88] G. Brassard, D. Chaum, and C. Crépeau. Minimum disclosure proofs of knowledge. *JCSS*, 37(2):156–189, 1988.

[Blu82] M. Blum. Coin flipping by telephone. In *Proc. IEEE Spring COMPCOM*, pages 133–137. IEEE, 1982.

[BM81] M. Blum and S. Micali. Coin flipping into a well. Unpublished, 1981.

[BM84] M. Blum and S. Micali. How to generate cryptographically strong sequences of pseudo-random bits. *SIAM J. Computing*, 13(4):850–863, November 1984.

[BMO90] M. Bellare, S. Micali, and R. Ostrovsky. The (true) complexity of statistical zero-knowledge. In *Proc. 22nd ACM Symposium on Theory of Computing*, pages 494–502, Baltimore, Maryland, 1990. ACM.

[Dam88] I.B. Damgård. Collision free hash functions and public key signature schemes. In David Chaum and Wyn L. Price, editors, *Proceedings of EUROCRYPT 87*, pages 203–216. Springer-Verlag, 1988. Lecture Notes in Computer Science No. 304.

[Dam90] I.B. Damgård. On the existence of a bit commitment schemes and zero-knowledge proofs. In G. Brassard, editor, *Proceedings CRYPTO 89*, pages 17–29. Springer-Verlag, 1990. Lecture Notes in Computer Science No. 435.

[DPP94] I.B. Damgård, T.P. Pedersen, and B. Pfitzmann. On the existence of statistically hiding bit commitment schemes and fail-stop signatures. In Douglas R. Stinson, editor, *Proceedings CRYPTO 93*, pages 250–265. Springer, 1994. Lecture Notes in Computer Science No. 773.

[GM84] S. Goldwasser and S. Micali. Probabilistic encryption. *JCSS*, 28(2):270–299, April 1984.

[GMR88] S. Goldwasser, S. Micali, and R. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM J. Computing*, 17(2):281–308, April 1988.

[GMW91] O. Goldreich, S. Micali, and A. Wigderson. Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems. *Journal of the ACM*, 38(1):691–729, 1991.

[IOS94] T. Itoh, Y. Ohta, and H. Shizuya. Language dependent secure bit commitment. In Yvo G. Desmedt, editor, *Proceedings CRYPTO 94*, pages 188–201. Springer, 1994. Lecture Notes in Computer Science No. 839.

[Nao90] M. Naor. Bit commitment using pseudo-randomness. In G. Brassard, editor, *Proceedings CRYPTO 89*, pages 128–137. Springer-Verlag, 1990. Lecture Notes in Computer Science No. 435.

[NOVY92] M. Naor, R. Ostrovsky, R. Venkatesan, and M. Yung. Perfect zero-knowledge arguments for *np* can be based on general complexity assumptions. In Ernest F. Brickell, editor, *Proceedings CRYPTO 92*, pages 196–214. Springer-Verlag, 1992. Lecture Notes in Computer Science No. 740.

A A Proof of Lemma 3

Before we can prove Lemma 3 we need to develop some number-theoretic tools and notations.

A.1 Tags of Elements Modulo Prime-Powers

Let $p = q^e$ be an odd prime-power (that is, q is an odd prime and e is a positive integer). Denote by $\phi(p)$ the order of the multiplicative group Z_p^* and denote by m the largest integer such that 2^m divides $\phi(p)$. Also, let g be some "canonical" generator in Z_p^* (e.g. the smallest generator in Z_p^*).

Definition 4. Let z be an element in Z_p^* and let ℓ be the discrete log of z base g. (i.e., $g^\ell \equiv z \bmod p$). The *tag of* $z \bmod p$ is the residue of ℓ mod 2^m. We denote it by $TAG_p(z)$. That is,

$$\text{for all } 0 \leq \ell < \phi(p),\ TAG_p(g^\ell) = [\ell]_{2^m}$$

where $[x]_y$ denotes the residue of x mod y. The following properties are immediate from the definition of the tag:

Claim 5. *Let p be an odd prime power and m be the largest integer such that 2^m divides $\phi(p)$.*

1. *If* $w, x, y, z \in Z_p^*$ *such that* $TAG_p(w) = TAG_p(x)$, $TAG_p(y) = TAG_p(z)$ *then* $TAG_p(wy) = TAG_p(xz)$.
2. *An element* $z \in Z_p^*$ *is a square mod* p *iff it has an even tag.*
3. *If* $TAG_p(z) = 0$ *then so is* $TAG_p(z^2)$.
4. *For any* $z \in Z_p^*$ *and any* $i \geq m$, $TAG_p(z^{2^i}) = 0$.
5. *Let* z *be a square mod* p *and denote its tag by* $TAG_p(z) = 2^i r$ *where* r *is some odd integer and* $1 \leq i \leq m$. *Then, one square root of* z *has tag* $2^{i-1}r$ *and the other has tag* $\left[2^{i-1}r + 2^{m-1}\right]_{2^m}$.

The following corollaries describe the behavior of the tags under the functions $f_{p,s}$ (and their inverses):

Corollary 6. *If* p *is an odd prime-power,* x, y *are elements in* Z_p^* *such that* $TAG_p(x) = TAG_p(y)$ *and* s *is any string, then* $TAG_p(\, f_{p,s}(x)\,) = TAG_p(\, f_{p,s}(y)\,)$.

Corollary 7. *If* p *is an odd prime-power and* z *is an element in* Z_p^* *then the tags of the pre-images of* z *under both* $f_{p,0}(\cdot)$ *and* $f_{p,1}(\cdot)$ *(if there are any) depend only on the tag of* z.

Corollary 8. *If* p *is a prime power,* $p < 2^k$, *then for any element* $x \in Z_p^*$, *any string* s *and any* $i \geq k$, $TAG_p\left(f_{p,0^i s}(x)\right) = TAG_p\left(f_{p,s}(x)^{2^i}\right) = 0$

The following is the main technical lemma in the proof

Lemma 9. *Let* p *be an odd prime-power and let* $y, z \in Z_p^*$, *so that* $TAG_p(y) = TAG_p(z)$. *Then for any string* s *we have*

$$\#\{x \in Z_p^* \;:\; f_{p,s}(x) = y\} \;=\; \#\{x \in Z_p^* \;:\; f_{p,s}(x) = z\}$$

where $\#A$ *denotes the number of elements in the set* A.

Proof. Let $s = b_0 \cdots b_{n-1}$ be a string and consider the "pre-images-tree" w.r.t. s that is rooted at an element z (i.e., the children of z are its pre-images under f_{p,b_0}, their children are their pre-images under f_{p,b_1} etc.). An element in this tree is an internal node if it is a square mod p and its distance form the root is less than n. Otherwise, it is a leaf.

Notice that this is indeed a tree in the sense that all the elements at distance i from the root are distinct. We will now show that if y and z have the same tag then their trees w.r.t. s are isomorphic. This means, in particular, that the number of elements at distance n from the root is the same in both trees, which implies Lemma 9. The notion of pre-images-tree becomes formal in the following definition:

Definition 10. Let p be an odd prime power, let $z \in Z_p^*$ and let $s = b_0 \cdots b_{n-1} \in \{0,1\}^n$. The *pre-images-tree w.r.t.* s *which is rooted at* z is a directed graph $T_{z,s} = (V, E)$ where

$$V = \{\langle i, x\rangle \;:\; 0 \leq i < n,\; x \in Z_p^*, \text{ and } f_{p,b_0 \cdots b_{i-1}}(x) = z\}$$
$$E = \{\langle i, y\rangle \rightarrow \langle i+1, x\rangle \;:\; y = f_{p,b_i}(x)\}$$

To see that this is a directed rooted tree, notice that $\langle 0, z\rangle$ has in-degree 0, every other node has in-degree 1, and there is a path from $\langle 0, z\rangle$ to every other node in the graph. Now we can prove Lemma 9 by proving a stronger lemma

Lemma 11. *For every string s and every two elements $y, z \in Z_p^*$ such that $TAG_p(y) = TAG_p(z)$, there is an isomorphism between $T_{y,s}$ and $T_{z,s}$ which also preserves the tags. That is, there exists a function $I : T_{y,s} \to T_{z,s}$ which satisfies the following properties:*

1. *I is an isomorphism between the graphs $T_{y,s}$ and $T_{z,s}$ (notice that this implies that I always maps nodes in level i in $T_{y,s}$ to nodes in level i in $T_{z,s}$).*
2. *If $I(\langle i, x\rangle) = \langle i, x'\rangle$ then $TAG_p(x) = TAG_p(x')$.*

In particular, it follows that the number of nodes in level n in both trees is the same, which implies Lemma 9.

Proof. The proof is by induction over n (the number of bits in s). It consists of a straightforward implementation of Corollary 7 above. *details omitted.* □

A.2 Tags of Elements Modulo Composites

Definition 12. Let N be an odd integer, and let $z \in Z_N^*$. Denote N's prime factorization by $N = p_1 \cdots p_\ell$ where $p_1, \cdots, p_\ell$ are powers of distinct primes. The *tag of z mod N* is the vector $\langle t_1, \cdots, t_\ell\rangle$ where t_i is the tag of z mod p_i.

Notice that for an element $x \in Z_N^*$ and a string s we have $[f_{N,s}(x)]_{p_i} = f_{p_i,s}([x]_{p_i})$ for all i. Therefore, from Corollary 8 we get

Corollary 13. *If $N < 2^k$, then for any element $x \in Z_N^*$, any string s and any $i \geq k$ we have $TAG_N\left(f_{N,0^i s}(x)\right) = \langle 0, 0, \cdots, 0\rangle$.*

and from Lemma 9 we get

Claim 14. *Let N be an odd integer and let $y, z \in Z_N^*$, so that $TAG_N(y) = TAG_N(z)$. Then for any string s we have*

$$\#\{x \in Z_N^* \ : \ f_{N,s}(x) = y\} \ = \ \#\{x \in Z_N^* \ : \ f_{N,s}(x) = z\}$$

Proof. Follows since for all $y \in Z_N^*$

$$\#\{x \in Z_N^* \ : \ f_{N,s}(x) = y\} \ = \ \prod_i \#\{x \in Z_{p_i}^* \ : \ f_{p_i,s}([x]_{p_i}) = ([y]_{p_i})\}$$

□

From Corollary 13 and Claim 14 we get

Lemma 15. *Let N be an odd k-bit integer, and let s be any string. Then, for every element $y \in Z_N^*$ with tag $\langle 0, \cdots, 0\rangle$ we have*

$$\#\{x \in Z_N^* \ : \ f_{N,0^k s}(x^2) = y\} \ = \ \frac{\phi(N)}{T_0}$$

where $\phi(N)$ is the order Z_N^* and T_0 is the number of elements in Z_N^* with tags $\langle 0, \cdots, 0\rangle$.

Proof. From Corollary 13 we know that $f_{N,0^k s}(x^2)$ has tag $\langle 0,\cdots,0\rangle$ for all $x \in Z_N^*$. Therefore the pre-images of all the y's with tag $\langle 0,\cdots,0\rangle$ cover all Z_N^*. From Claim 14 we know that all these pre-images have the same size, which mean that this size is exactly $\phi(N)/T_0$ (we can apply Claim 14 since $f_{N,0^k s}(x^2) = f_{N,0^k s0}(x)$).

□

A.3 Back to Lemma 3

Recall the experiment of Lemma 3. Alice has an k-bit odd integer N. She picks a string $\sigma \in \{0,1\}^*$ according to some distribution D, computes $s = Enc(\sigma)$, picks at random an element $x \in Z_N^*$, and computes $y = f_{N,0^k s}(x^2)$.

We denoted by S, X the random variables which take on the values of s, x respectively. The lemma asserts that the value of y does not give any information about s. Since Alice picks x randomly in Z_N^* regardless of s, then S and X are independent. Therefore, from Lemma 15 we get

Corollary 16. *For any element y such that $TAG_N(y) = \langle 0,\cdots,0\rangle$ and any string s*

$$\Pr_{X,S}\left[S = s \text{ and } f_{N,0^k S}(X^2) = y\right]$$
$$= \Pr_S[S = s] \cdot \Pr_X\left[X \in \{x \in Z_N^* \ : \ f_{N,0^k s}(x^2) = y\}\right] \ = \ \Pr_S[S = s] \cdot \frac{1}{T_0}$$

Notice also that if $TAG_N(y) \neq \langle 0,\cdots,0\rangle$ then $\Pr_{X,S}\left[f_{N,0^k S}(X^2) = y\right] = 0$. This implies that for every y for which $\Pr_{S,X}\left[f_{N,0^k S}(X^2) = y\right] > 0$ we have

$$\Pr_{X,S}\left[S = s \mid f_{N,0^k S}(X^2) = y\right] \ = \ \Pr_S[S = s]$$

which is what we need. □