Sha-zam: A Block Cipher. Fast as DES, Secure as SHA

Sarvar Patel
Lucent Technologies
67 Whippany Road
Whippany, NJ 07981
sarvar@lucent.com

Zulfikar Ramzan*
Laboratory for Computer Science
Massachusetts Institute of Technology
Cambridge, MA 02139
zulfikar@theory.lcs.mit.edu

Ganesh Sundaram
Lucent Technologies
67 Whippany Road
Whippany, NJ 07981
ganeshs@lucent.com

December 6, 1999

Abstract

We describe a block cipher which is both practical and provably secure as SHA-1. The cipher uses the Secure Hash Algorithm (SHA-1) as an underlying primitive, and we show that any succesful attack on the cipher results in a succesful attack against one or more of the hallowed properties of SHA-1. Moreover, our block cipher is still as fast as the Data Encryption Standard (DES). We also describe a practical Pseudo-Random Generator which again is as secure as SHA-1. We apply this generator for secure key scheduling and since it is based on the same underlying primitive as our cipher, we get efficient reuse of our code. Finally we describe a construction of an efficient family of universal hash functions which are used by our cipher, which may be of independent interest.

1 Introduction

Designing practical yet provably secure cryptographic algorithms is one of the greatest challenges in modern cryptography. In fact the notions of practicality and provable security have often seemed contradictory. Usually provably secure

*Work done while this author was at Lucent Technologies

systems rely on heavy duty mathematical tools which often end up slowing the system down immensely. On the other hand, one can attempt to design systems in a fast, ad-hoc manner, but usually it's difficult to make any statements about the security of these systems, and the upshot is that most such systems end up being broken.

In this paper we propose the design of a practical but yet provably secure block cipher based on SHA[1] as one of the underlying building blocks of our system. We combine SHA with what's called a *Feistel Permutation,* and a *square hash function (SQH).* This permutation function is one of the design principles that is central to the data encryption standard (DES), whereas SQH is fundamentally new. In this work we present the security results (and refer to the proofs published elsewhere) under some reasonable and accepted assumptions about SHA, that our block cipher is secure. The novelty in our proof is that we show different levels of security under different assumptions about SHA. Under some of the stronger assumptions about SHA we can make very strong guarantees of security, and under certain weaker assumptions about SHA we can make weaker guarantees of security. This enables us to avoid "putting all of our eggs in one basket." This way even if many years down the line, someone shows that a particular assumption about SHA is false (though this is commonly believed to be unlikely) our system may still be secure. Moreover, our system is designed such that it might still be a secure system even if all the underlying hallowed assumptions about SHA are wrong (which is, once again, believed to be unlikely).

The central idea in our construction is based on a result of Luby and Rackoff, [LR], where they use Feistel networks to construct pseudo random permutations from pseudo random functions. In particular they have outlined the construction of a block cipher from pseudo random functions using Feistel Networks. This work of Luby and Rackoff has initiated an explosion of research on applications and generalizations. More recently, Maurer [M] has greatly simplified the treatment and proofs of [LR], and Lucks [L] has improved the efficiency using *difference concentration.* This work was followed by a more efficient construction by Naor and Reingold, [NR1], where they partially replace the Feistel permutation by universal hash functions. This modification improves the efficiency with no compromise on security. Furthermore, they present proofs for a four round construction which is secure against adaptive plaintext and adaptive ciphertext attacks.

Replacing pseudo random functions (PRF) by cryptographic functions (with desired properties) is not new. Anderson and Biham [AB], propose the use of Feistel networks in conjunction with stream ciphers to design block ciphers. Also, Lucks used MD5 with an unbalanced Feistel network and Guttman's construction uses SHA but different from the Luby Rackoff construction.

[1]- To be precise we use SHA-1 in our construction. But for brevity we will refer to it as SHA in the entire paper.

In our design we do not use any stream ciphers. Instead we rely entirely on the improved versions of the Luby Rackoff construction and use SHA as our underlying primitive. In addition to this, we use another building block called a Δ-universal hash function. We propose a new family of such hash functions and we prove that this family of functions has the desired properties. This use of hash functions, similar to the Naor-Reingold construction, makes our cipher secure against adaptive attacks (as mentioned earlier). Notably, our construction is simpler compared to the Naor-Reingold scheme since we use a triplet of keys for four rounds, and the hash functions we utilize in the first and fourth rounds can be implemented more efficiently than the ones originally proposed by Naor and Reingold. This simplification and optimization comes with no compromise of security, and we prove this fact within.

The novelty of this paper lies in the fact that our design is practical but yet provably secure. We use familiar constructions from contemporary cryptography. The contributions of this work lie in the following: We introduce a new family of Δ-universal hash functions which aids in the efficiency of the overall system. In fact, this family of hash functions is faster than contemporary families with similar properties used in cryptography [EPR]. In addition, properties specific to this family of hash functions allows us to use three keys for four rounds thereby increasing the efficiency. Next, our use of the Feistel permutation is slightly modified without any sacrifice in the security. This modification was motivated by the observation that the core of the proof of Luby and Rackoff regarding the properties of the Feistel permutation relies on the fact that the operations are being performed in a group. In other words, there is nothing specific about the XOR operation (as in the conventional Feistel permutation). In our design we use the group of integers modulo 2^n instead of the Galois field $GF(2^n)$ and prove that this modification does not introduce any weaknesses. Additionally, the triplet of keys which drive the cipher are generated by a provably secure pseudo random bit generator which also uses SHA as the underlying primitive. This pseudo random bit generator, whose security is related to SHA, runs at approximately half the speed of the overall cipher but every bit of the output is individually hard. Since we use SHA for this generator also, we get very efficient reuse of code, which helps in improving the overall efficiency of the system. The proofs of security, are of two kinds: First under the assumption that SHA provides us with a family of pseudo random functions, we show our system is secure. Next under the "message authentication code (MAC)" assumption (which is weaker) we show our cipher is secure. This assumption is very practical, and in fact is at the heart of many systems currently in practice. A good example is the HMAC construction. In fact, even if HMAC is broken our system might be secure. This second proof (where we require the IV to be secret) based on a weaker assumption, for a Luby Rackoff cipher, is fundamentally new. Finally, we observe that the overall system is as fast as DES (the data encryption standard). Note that, DES has been a world wide standard and by far the most popular encryption algorithm. The security of DES (although

under debate in recent times) is believed to be high and the efficiency of the encryption and decryption processes are very fast in most microprocessors.

The assumptions we make about SHA are widely believed to be true. In addition it is both a NIST and internet standard. So the claims of provable security we make are based on the *assumption* that SHA has all these hallowed properties. In addition, under one of the assumptions (i.e., Secure MAC assumption) we do not make any claims on partial leakage of bits of a message. This we consider to be a theoretical weaknesses in the system, but should not be of much concern in practice.

The rest of the paper is organized as follows. We start with some background information on practical versus provably secure cryptography. This is followed by a discussion of the basic tools that are relevant to our construction in section 3. Section 4 provides a description of the block cipher, and in section 5 we discuss performance issues. This is followed by the key scheduling algorithm in section 6, and in section 7 we discuss the proofs of the security of the system. A few of the design choices we made, which seemingly improve the security of the system, are discussed in section 8.

2 Background

In the design of unconditionally secure cryptosystems, the goal is to show that it is *absolutely infeasible* for any adversary, no matter how clever, to break the cryptosystem. But, *no one* to this day has been able to design an unconditionally provably secure cryptosystem (with one notable, but very impractical exception: the one-time pad). The approach taken by most people trying to design provably secure systems is to start with some type of basic *building block* or *primitive*. Then, you make some reasonable assumptions, which you may not be able to formally prove, but yet justify for all practical purposes, about this primitive. Usually such assumptions should be well accepted and widely believed, even though no formal proof is known. Finally, you show that any attack which breaks the security of your new cryptosystem will cause one of the hallowed assumptions you made on the underlying primitive to be broken. Since you assumed, and it is widely believed, that the assumption on the underlying primitive is valid, it then becomes unlikely that the cryptosystem can be broken. This approach was pioneered in a seminal paper by Blum and Micali [BM], where they introduced the notion of a secure pseudo random number generator based on complexity theoretic assumptions.

One problem with this approach is that often, in order to design systems like the ones mentioned above, one needs a great deal of heavy duty mathematical machinery, and this can often cost you a great deal of efficiency. For example, many people have used famous unsolved problems in number theory as the underlying primitives. Unfortunately, most number theoretic notions are very expensive to implement on current microprocessors. Some of the more popular

number theoretic problems considered are the discrete log problem in a finite field and the problem of factoing integers. The reader is referred to [PS] for a survey of the work on discrete log based generators and the design of an efficient generator. The benefit of these systems is that, although they are very slow, we are almost guaranteed that no one can break them.

The design of practical cryptosystems is in many ways an art. The goal is to use inexpensive operations, and be clever enough so that no one can break the system. Usually cleverness entails doing enough bit manipulation so as to confuse any adversary thereby preventing breakage of the cryptosystem. The nice thing about this approach is that the resulting systems are often very fast. Unfortunately, the security of the cryptosystems rests almost solely on the cleverness of the person who designed it. There are several well known systems that were designed to be very efficient, but for which there are no known mathematical proofs of security. *Examples: SHA-1, DES.* A few of these systems have "withstood the test of time" and have still not been broken. One of the newest trends in cryptography has been to study these particular systems, and understand what it is about them that makes them secure. And to see if these systems in some form can be incorporated and used as the building blocks for other systems. This brings us to the forefront of current research, which involves the design and analysis of practical yet provably secure cryptosystems.

Perhaps the main approach to designing practical, yet provably secure cryptosystems has been to use fundamental building blocks which are both fast, and for which there are many hallowed properties which are believed and widely accepted to be true. One example of such a building block is the Secure Hash Algorithm (SHA). SHA is believed to have several very nice properties. Unfortunately, no one has been able to prove that these properties hold, but it appears that they do. Moreover, to this day, no one has been able to show that any of these believed properties are false – though many have tried.

3 Basic Tools

The Sha-zam cipher makes use of two building blocks. The first of these is the Secure Hash Algorithm (SHA) designed by NIST along with the NSA. The second of these basic building blocks is the Square Hash (SQH) which we have developed. We describe the construction of SHA and SQH in the subsequent sections and discuss their relevant properties. Before proceeding further, we discuss the relevant notation.

3.1 Notation

Given two strings x and y we denote by (x, y) their concatenation. Also we denote by $\{0,1\}^k$ the set of all bit strings of length k. If C is an n-bit string, we denote by $prefix_k(C)$ the $n-k$ bit prefix of C (i.e. the first $n-k$ bits of C).

3.2 The Secure Hash Algorithm (SHA)

The Secure Hash Algorithm (SHA) was designed by the National Institute of Standards and Technology (NIST) along with the NSA (National Security Agency) to be used with the Digital Signature Standard. SHA was modled closely after the MD family of message digest algorithms devloped by Rivest. SHA takes a 512 bit input and produces a 160 bit output. SHA also has a 160 bit Initialization Vector (IV) which can be modified but there is a standard setting for this vector which is believed to give good security. SHA was designed to make the process of digitally signing messages more practical. In particular the idea is that instead of signing the entire message, you first apply SHA to the message, get an output of shorter length than the input, and then sign this shorter value – which would take less time than signing the original larger message.

We denote by $SHA(IV, x)$ the 160 bit output produced by SHA on a 512 bit user specified input x and the standard IV. SHA is believed to have several fundamental properties which make it an excellent building block for cryptographic protocols and algorithms:

1. SHA behaves like a random function: If some significant portion of the input is kept secret, then there is no computationally feasible mechanism for correlating the remaining input bits with the output bits of SHA.

2. SHA acts as a secure Message Authentication Code: If there is a relatively large (greater than say 128 bits) secret value s, then any adversary who gets to see pairs $(m_1, t_1), \ldots (m_n, t_n)$ where each $t_i = SHA(IV, m_i, s)$, will not be able to come up with a pair (m', t') where $t' = SHA(IV, m', s)$ and where m' is different from each of the m_i's. Moreover, the adversary will be unsuccesful even if this attack is mounted *adaptively*; e.g. the adversary can pick a message m_1, be given the corresponding tag $t_1 = SHA(IV, m_1, s)$, and from this information can pick m_2, see the corresponding t_2 and so on – it will still be impossible for the adversary to come up with a valid (m', t') pair where m' is different from the other m_i.

These are some of the accepted and believed properties of SHA. To this day, no one has been able to violate even a single one of these hallowed properties. We show that *any* attack on the Sha-zam block cipher will violate at least one of these properties. In fact, we can show that in order to "completely" break the Sha-zam cipher and the key scheduling generator, one would have to violate *all* of the above mentioned properties. Moreover, the converse is not necessarily true. That is, someone can violate all three properties (which is highly unlikely) and still not be able to break our cipher.

3.2.1 Assumptions related to Sha-zam

In our proofs of security of Sha-zam, we make the following assumptions:

1. keyed variant of SHA-1 is pseudo-random
2. keyed variant of SHA-1 is a secure MAC

Assumption 1 is stronger and, in fact, assumption 1 implies assumption 2. Indeed, Sha-zam attains the highest possible level of security of strong pseudo-randomness, under the first assumption. In addition, we prove security against inversion and forging type attacks under the second assumption, i.e., that the keyed variant of SHA-1 is a secure Message Authentication Code.

3.2.2 Evidence

The pseudo-random function assumption on SHA-1 can be justified as follows.

1. This assumption follows, for example, from the random oracle assumption on SHA-1. Intuitively, this means that it is infeasible for any adversary to establish any kind of corellation between the input bits and the output bits. In other words, the output bits appear to be "random." Several proofs of security for various protocols in cryptography rely on the random oracle model. In practice a vast majority of them use SHA-1 as a random oracle, and in particular none of these protocols which use SHA-1 and whose proof of security is based on the random oracle model have been broken. In other words implicitly various protocols already assume that SHA-1 behaves like a random oracle (which is a stronger assumption than the assumption that SHA-1 behaves like a pseudo-random function).

2. In addition, although SHA-1's immediate design goals do not discuss pseudo-randomness, it is well accepted that in order for SHA-1 to meet its intended design goal (collision-intractability) it should have good statistical properties and computational properties such as pseudo-randomness. For instance the designers of RIPEMD in their detailed design note [BP] state:

   "It is the general view that in order to avoid possible statistical attacks, a good cryptographic hash function should behave like a random function. The same holds for the compression function....

   Statistical irregularities in the behavior of this function could be used in finding collisions or messages that produce a given hash code."

   Furthermore, they state that (based on statistical tests):

   "We can conclude from the results of these tests that there is no indication that the statistical behavior of the function compress *used in RIPEMD differs significantly from what can be expected for a random function."*

   The last quotation, applies to RIPEMD but it is widely believed that SHA-1 and RIPEMD have similar strengths.

3. Another instance, where the pseudo-randomness assumption is used is in the *pseudo-random number generator for DSS*. This is a NIST standard [DSS]. This generator cannot be cryptographically secure unless we make the assumption that SHA-1 behaves like a pseudo-random function. For details the reader is referred to [MvV].

4. Other examples of ciphers which rely on the assumption that SHA-1 and/or MD5 behave like pseudo-random functions can be found in the works of [AB] and [Lu]. In particular, the latter cipher by Lucks involves a Luby-Rackoff type construction albeit with an unbalanced Feistel network.

Even if the pseudo-random assumption is not to be trusted, we have another proof of security based on the weaker message authentication code (MAC) assumption. The MAC assumption is universally believed.

1. It is used in the popular HMAC construction as noted in the Internet RFC [HMAC]. For the details of the proof of the HMAC construction the reader is referred to [BCK].

2. Another instance is the use of SHA-1 in challenge response protocols. Recall that these authentication protocols generally rely on the MAC property of SHA-1. Hence in particular, if the MAC assumption on SHA-1 was not true then the system would have been compromised at the authentication level to begin with.

3.2.3 Consequences

1. The assumption that SHA-1 behaves like a PRF allows one to make a very strong claim about Sha-zam: any attempt to "distinguish" will need on the order of 2^{80} plaintext-ciphertext pairs. Recall that, "distinguish" refers to any *test* which just discovers any difference between random and pseudo-random. Clearly it is harder to recover the keys than it is to distinguish.

2. Notice what we have proved provides us with *lower bounds* on security and the ACTUAL "security", in the conventional sense of key recovery, of Sha-zam may be much higher.

3. Another aspect of our construction is that *even if some compromises on SHA-1 with respect to the properties of interest are found, Sha-zam may yet be secure.* In other words, Sha-zam could very well be stronger than SHA-1.

Consequently little, if any, heuristic analysis needs to be done on Sha-zam; especially since its underlying building block, SHA-1, has been around for several

years. A lot of work on understanding the properties of SHA-1 has been done and at some point if weaknesses had been found then clearly we would not have used the pseudo-randomness assumption in our design.

3.3 The Square Hash

In this section we discuss the second fundamental building block of our block cipher: The Square Hash (SQH). SQH is what's called a Δ-universal family of hash functions. It has some very nice statistical properties, and is very easy to compute and evaluate. Because of its nice statistical properties, we can use Δ-universal hash functions in our block cipher to enable us to get a great degree of security at a very small cost. We start by defining the $SQH1$ family which gives the basic idea, and later we show how to modify this idea in order to make implementations easier. Here is the definition of $SQH1$:

Definition 1 *Let p be a prime. Define the $SQH1$ family of functions from Z_p to Z_p as:*

$$SQH1 \equiv \{h_x : Z_p \longrightarrow Z_p | x \in Z_p\}$$

where the functions h_x are defined as:

$$h_x(m) \equiv (m + x)^2 \bmod p$$

We now define what it means to be a Δ-universal family of hash functions:

Definition 2 *Let R be an Abelian Group and let $'-'$ denote the subtraction operation with respect to this group. Then H is a Δ-universal-family of hash functions if for all $x, y \in D$ with $x \neq y$, and all $a \in R$, $Pr[h(x) - h(y) = a] \leq 1/|R|$. H is called $\epsilon - almost - \Delta - universal$ if $Pr[h(x) - h(y) = a] \leq \epsilon$.*

Theorem 1 *The family $SQH1$ is a Δ-univeral family of hash functions.*

Here is the definition of SQH which is a minor modification on $SQH1$ to make implementations easier:

Definition 3 *Let l be a positive integer, and let p be a prime with $2^l < p < 2^l + 2^{l-1}$. The SQH family of functions from Z_p^k to Z_p is defined as follows:*

$$SQH \equiv \{g_x : \{0,1\}^l \longrightarrow \{0,1\}^l \mid x \in \{0,1\}^l\}$$

where the functions g_x are defined for any $x, m \in \{0,1\}^l$,

$$g_x(m) = (m + x)^2 \bmod p \bmod 2^l$$

Usually we denote by $SQH_k(m)$ the value $g_k(m) = (m + k)^2 \bmod p \bmod 2^l$.

When implementing the SQH family we usually take $l = 160$ and p to be the smallest prime number greater than 2^{160}. We now state our main theorem regarding the statistical properties of the SQH family:

Theorem 2 *SQH is an ϵ-almost-Δ-universal family of hash functions with $\epsilon \leq 3 \cdot 2^{-l}$*

Since we typically take l to be a large quantity (a 160 bit number), $3 \cdot 2^{-l}$ is a negligible quantity.

The proofs of theorems 1 and 2 are provided in, [EPR].

4 Description of Block Cipher

Our block cipher, which we call Sha-zam, takes as input a 320 bit block M and outputs a 320 bit ciphertext C. We denote $M = (L, R)$ where L is the left 160 bits of M and R is the right 160 bits of M. Also, we prefer to keep IV secret.

Encryption with Sha-zam

Input:	Plaintext Stored in L, R – each of which is 160 bits Private Key: $k = (k_1, k_2, k_3)$ where: k_1, k_3 are 160 bits each, and k_2 is 352 bits. If IV not secret: then use the standard 160 bit IV.
Output:	Ciphertext stored in V,W – each of which is 160 bits
Procedure:	$S = L + SQH_{k1}(R) \bmod 2^{160}$ $T = R + SHA(IV, S, k_2) \bmod 2^{160}$ $V = S + SHA(IV, T, k_2) \bmod 2^{160}$ $W = T + SQH_{k_3}(V) \bmod 2^{160}$

Decryption with Sha-zam

Input:	Ciphertext Stored in V,W – each of which is 160 bits Private Key: $k = (k_1, k_2, k_3)$ where: k_1, k_3 are 160 bits each, and k_2 is 352 bits. If IV note secret: then use the standard 160 bit IV.
Output:	Plaintext stored in L,R
Procedure:	$T = W - SQH_{k_3}(V) \bmod 2^{160}$ $S = V - SHA(IV, T, k_2) \bmod 2^{160}$ $R = T - SHA(IV, S, k_2) \bmod 2^{160}$ $L = S - SQH_{k_1}(R) \bmod 2^{160}$

5 Performance of Sha-zam

Our block cipher can be implemented efficiently. Specifically, it can encrypt messages in roughly the same time as it would take DES to accomplish this same task. We start by comparing the processing speed (in megabits / second) of optimized SHA and DES implementations. On a Pentium 100 running Linux, SHA works at 18.6 megabits per second, whereas DES produces 17.864 megabits per second. On a Pentium Pro 200 running Win 95, SHA works at 39.656 megabits per second, and DES works at 37.322 megabits per second. Here is a table summarizing these results:

Platform	*SHA*	*DES*
Pentium 100 Running Linux	18.6 mbits/sec	17.864 mbits/sec
Pentium Pro 200 Running Win 95	39.656 mbits/sec	37.232 mbits/sec

These results were derived using the SSLay Optimized Assembly implementations of SHA and DES. This implementation can be found at: ftp://ftp.psy.uq.oz.au/pub/Crypto/libeay/.

For our implementation of Sha-zam we can actually speed things up because part of our input to SHA is held to be a fixed constant. This should give us approximately a 10% speedup for each of the two calls to SHA. Moreoever, we also believe that each call to SQH can be implemented within this 10% window [EPR]. We derived at this conclusion by analyzing the instruction counts of an optimized assembly language implementation of SQH. Therefore, we can implement our entire cipher in the same amount of time as DES.

Note that, these implementation results are preliminary. More work needs to be done to confirm these numbers and it is possible to improve some of these timings.

6 Key Scheduling

We describe a practical and provably secure pseudo-random generator based on SHA. If we run our generator using a randomly selected 100-bit key as an input seed, we can securely generate the necessary 672 bits needed for the secret key of our block cipher. If the initial value for SHA (i.e., IV) is also a secret, then we will generate an extra 160 bits, for a total of 832 bits. We can also run our generator assuming a smaller or larger initial seed and get an appropriate level of security – so the system is flexible. The proofs of security of our generator, which relies on the security of SHA, will appear elsewhere.

6.1 Description of Generator

We start by describing some of the relevant parameters and building blocks of our generator. Our generator makes use of several parameters. We make use

of a 512 bit prespecified global constant C. We almost never use the entire constant C but often take some specified prefix of it depending on the length of the key we're working with. We now describe our generator. Given a secret key s and a counter c we generate random bits as follows:

Description of Secure Pseudo Random Number Generator
Input: secret key s and counter c

1. Load the payload registers of SHA with prespecified global constant C.
2. Load the initial value IV with the standard SHA IV constant.
3. Let s' = the secret key XORed with the most signigicant bits of IV.
4. Let C'_i = the i-th counter value XORed with all the even words of the payload. 0-th is the least significant word, and 15-th is the most significant.
5. For i= 1 to m do
   Let $s_i = SHA(s', C'_i)$
6. Output: $\langle h(s_1), \ldots, h(s_m) \rangle$

In step 6 above, h refers to a hash function chosen from a universal class. For example, the linear congruential hash function is any finite field is a very good candidate. For our purposes we will be using linear congruential hash functions in a finite field of order p, where p is prespecified 161-bit prime number. Several optimal implementations of this hash function (which involves a multiplication and an addition in a finite field of the given order) exist.

7 Security of Sha-zam

In this section we discuss the security of the system. We are able to prove that Sha-zam is at least as secure as the underlying cryptographic primitive, SHA. In fact, under some reasonable assumptions about SHA we can show that an adversary (who has no knowledge of the secret key) *cannot even distinguish the output of Sha-zam from a purely random string.* This is the strongest possible claim of security you can possibly make. Under different assumptions about the security of SHA, we can show different levels of security for our block cipher. In the subsequent sections we describe the various levels of provable security we can achieve.

7.1 Security of the Feistel Construction

The overall security of the system relies on the Feistel construction, the proof of which was provided by Luby and Rackoff, and later simplied by Maurer [M]. Our design follows the Naor Reingold scheme [NR1], however, we improve upon their constructions in particular ways. First we use the same random function f

in rounds 2 and 3, but do not use full $2n$-bit pairwise independent permutations in rounds 1 and 4. Instead we use a n bit universal hash function in rounds 1 and 4. Furthermore the universal hash function does not have to be a strongly universal thus saving on key bits, but it is enough to be a certain type of *Delta*-universal hash function. Care needs to be taken here, for example the use of a linear delta-universal hash function will not guarantee a secure system. We show that the square hash function is sufficient. Square hash also has the added benefit of being twice as efficient as the linear hash. To be able to use square hash we have to also change the XORs between rounds to addition modulo 2^n. We refer the reader to [PRS1] for a proof of the following theorem.

Theorem 3 *Let h_1 and h_2 be SQH Δ-universal hash functions and f be a random function. The block cipher defined by a 4 round Feistel network where $f_1 = h_1$, $f_2 = f$, $f_3 = f$, and $f_4 = h_2$ is distinguishable from a random permutation with Prob $\leq O(\frac{m^2}{2^n})$ where m is the number of queries made by an adversary.*

7.2 Security Under the Secure Message Authentication Code Assumption

We consider the case when the adversary is able to decrypt ciphertext without knowing the secret key. We already know that under the pseudo-random assumptions on SHA this is very unlikely. In any case, we now give further evidence that this is unlikely under a slightly weaker but widely held assumption: That SHA can be used for Secure Message Authentication. This assumption is quite widely believed – in fact it were false, then the proofs of security in HMAC (which is an internet RFC) would not work. We start by defining some basic notions relating to Message Authentication Codes (MACs), (see [BCK]), and later show that in fact the task of decryption in Sha-zam without knowledge of the secret key is much more difficult than violating the MAC security of SHA. In this section, we will use Sha-zam with a secret initial value. In other words we will generate 160 extra bits out of the key scheduling generator (for a total of 832 bits) and allot these bits for the IV.

A Message Authentication Code (or MAC) is a function which has a secret key k and takes as input a message m. It returns as output a tag $MAC_k(m)$. MAC's are used by two people engaged in communication (who have agreed on a secret key) to ascertain that the messages they've sent across are in fact valid, authentic, and have not been tampered with. There is a stringent notion of security for MACs. Here is the most general type of attack that can occur. An adversary gets to see a sequence $(m_1, t_1), (m_2, t_2), \ldots (m_q, t_q)$ of (message, tag) pairs; i.e. $t_i = MAC_k(m_i)$. The adversary, however, is not told what the secret key k is. We say that the adversary breaks the MAC if he can come up with a message m which is different from each of the messages $m_1, \ldots, m_q$, and a corresponding tag t such that $t = MAC_k(m)$.

The (message, tag) pairs that the adversary gets to see could be derived in several different ways. It could be that the adversary managed to eavesdrop on some communication between two parties, and had no control over the messages for which he saw the tag. Or, the adversary could have been given restricted black box access to the MACing algorithm. Specifically, the adversary could have chosen messages $m_1, \ldots, m_q$ and been told the corresponding tags $t_1, \ldots, t_q$ such that $t_i = MAC_k(m_i)$. This type of attack is called a *chosen message attack.* Or, the worst possible scenario – the adversary was given unrestricted black box access to the MACing algorithm. Specificially, the adversary can choose m_1, get to see the corresponding tag t_1. From that information, the adversary can pick m_2, receive the corresponding tag t_2, and so on. In this case, the adversary is said to mount an *adaptive chosen message attack* because the adversary gets to adaptively choose his queries to the black box based on the answers to the previous queries.

We now formally state our assumption on SHA's ability to be a secure MAC:

Assumption 1 *Let k be a 352 bit secret key. Consider the MAC on a 160 bit message m defined by $MAC_k(m) = SHA(IV, m, k)$. We assume that it is computationally infeasible for an adversary to break the MAC (even under an adaptive chosen message attack).*

One thing to note is that the above assumption is actually much weaker than the standard assumptions made about SHA's ability to act as a good MAC. Normally when one thinks of a MAC, both the size of the secret key and the size of the output are much smaller than the size of the input message. Since we have a relatively large key and a relatively large output size, we get a much better level of security. We now state our main theorem regarding the security of our cipher under the assumption that SHA serves as a secure MAC. The proof of this theorem can be found in [PRS1].

Theorem 4 *Suppose there is an adversary A that is given black box access to both the encryption and decryption functions for Sha-zam. And after some number q of queries to the black box, is capable of either encrypting or decrypting a randomly chosen message M. Then this adversary A can be converted to another adversary (call it A') which can break the MAC denoted by: $MAC_k(m) = SHA(IV, m, k)$ by making $2q$ adaptive queries. (Thus violating the assumption that SHA serves as a secure MAC.)*

8 Miscellaneous design choices

In our design of Sha-zam, we chose to use Feistel networks involving addition over the group of integers modulo 2^n as opposed to conventional Feistel networks which rely on the XOR operation of bit strings. There are some deep scientific reasons that justify this new approach - in fact in a forthcoming paper [PRS2] we

show that constructing Luby-Rackoff ciphers over the group of integers modulo 2^n provides examples of ciphers which are secure which may not be secure involving conventional XOR operations. But that discussion is beyond the scope of this note. On the other hand let us observe heuristically what is going on - which led us to formalize what we did. Consider two bit strings A and B. When we XOR them then each bit of A is transformed into something by the corresponding bit of B. On the other hand, the operation of addition mod 2^n makes allowance for carry bits and so this allows for a "greater" mixing of bits. In other words a given bit of B can influence the output bits of multiple bits of A+B. A simple example of this phenomenon is as follows: consider the subset sum problem over the integers. This is known to be NP complete. But the same problem when considered over the set GF(2^n) is rather trivial to solve. In other words, conventional addition with carry bits makes it harder than XOR when addressed in the context of the subset sum problem.

Another design choice which we feel is noteworthy, is the use of square hash in the outer rounds instead of linear hash functions. Notice that square hash has the property of removing any linear dependencies between chosen inputs (as observed earlier). Overall, these choices (albeit heuristic in a certain sense) add a lot of value to the security (some of them provably) thereby providing some evidence to the claim that Sha-zam may yet be secure even if some hallowed property of SHA-1 is compromised.

9 Conclusion

We have described a practical and secure Block Cipher. We have shown that our cipher is as fast as DES, and is at least as secure as SHA. We have also described how to do secure key schedhuling via a practical and cryptographically secure pseudo-random bit generator where each bit is individually hard as SHA.

References

[ACGS] W. Alexi, B. Chor, O. Goldreich and C. P. Schnorr, RSA/Rabin bits are 1/2+1/poly(log N) secure, *Proceedings of 25th FOCS*, 449–457, 1984.

[AB] R. Anderson, E. Biham, Two practical and provably secure block ciphers, BEAR and LION, *Fast Software Encryption* (ed: D. Gollmann), LNCS 1039, 113–120, 1996.

[BCK] M. Bellare, R. Canetti, H. Krawczyk, Keying hash functions for message authentication, *Advances in Cryptology*, (ed: N. Koblitz), LNCS 1109, 1996.

[BCS] M. Ben-Or, B. Chor, A. Shamir, On the cryptographic security of single RSA bits, *Proceedings of 15th STOC*, 421–430, 1983.

[BBS] L. Blum, M. Blum, and M. Shub, A simple secure pseudo-random number generator, *SIAM J. Computing*, 15 No. 2:364–383, 1986.

[BM] M. Blum, and S. Micali, How to generate cryptographically strong sequences of pseudo random bits, *SIAM J. Computing*, 13 No. 4:850–864, 1984.

[BP] A. Bosselaers, B. Preneel, Integrity primitives for secure information systems, *RIPE-RACE 1040*, LNCS 1007, 1995.

[DES] U. S. Department of Commerce/ N. I. S. T, *Data Encryption Standard*, FIPS 46–2, 1977.

[DSS] U. S. Department of Commerce/ N. I. S. T, *Digital Signature Standard*, FIPS 186, May 1994.

[EPR] M. Etzel, S. Patel, Z. Ramzan, Square hash: Fast message authentication via optimized universal hash functions, *Advances in Cryptology*, LNCS 1666, 234–251, 1999.

[GL] O. Goldreich, and L. A. Levin, A hard-core predicate for all one way functions, *Proceedings of 21st STOC*, 25–32, 1989.

[GM] S. Goldwasser, and A. Micali, Probabilistic encryption, *Journal of Computer and Systems Science*, 28: 270–299, 1984.

[G] P. Gutmann, documentation to SFS release 1.20 - SFS7.DOC, URL:http:/www.cs.auckland.ac.nz/pgut01/sfs.html, 1995.

[HMAC] H. Krawczyk, M. Bellare, R. Canetti, HMAC: Keyed-Hashing for Message Authentication, *Internet RFC*, 2104, February 1997.

[L] L. Levin, Pseudorandom generators from one way functions, *Combinatorica*, 7(4): 357–363, 1987.

[LR] M. Luby, and C. Rackoff, How to construct pseudorandom permutations from pseudorandom functions, *SIAM Journal of Computing*, 17: #2, 373–386, 1988.

[Lu] S. Lucks, Faster Luby-Rackoff ciphers, *Proc. Fast Software Encryption*, Lecture Notes in Computer Science, vol. 1039, Springer-Verlag, 189–203, 1996.

[M] U. Maurer, A simplified and generalized treatment of Luby-Rackoff pseudorandom permutation generators, *Advances in Cryptology - EUROCRYPT '92*, LNCS 658, 239–255, 1992.

[MvV] A. Menezes, P. C. van Oorschot, and S. A. Vanstone, *Handbook of applied cryptography*, CRC press, 1997.

[NR1] M. Naor, O. Reingold, On the construction of pseudo-random permutations: Luby-Rackoff revis ited, *J. of Cryptology*, vol. 12, 29–66, 1999. Preliminary version in : *Proc. 29th Annual ACM STOC*, 189–199, 1997.

[NR2] M. Naor, O. Reingold, From unpredictability to indistinguishability: A sinple construction of pseudo random functions from MACs, *Advances in Cryptology*, LNCS 1462, 267–282, 1998.

[Na1] M. Naslund, Universal hash functions & hard core bits, *Proc. of Eurocrypt '95*, LNCS 921, 355–366, 1995.

[Na2] M. Naslund, All bits in $ax + b \bmod p$ are hard, *Proc. of Crypto '96*, LNCS 1109, 114–128, 1996.

[PRS1] S. Patel, Z. Ramzan, G. S. Sundaram, Towards making Luby-Rackoff ciphers optimal and practical, *Fast Software Encryption*, LNCS 1636, 171–185, 1999.

[PRS2] S. Patel, Z. Ramzan, G. S. Sundaram, Luby rackoff ciphers over arbitrary groups, *preprint*, 1999.

[PS] S. Patel, G. S. Sundaram, An efficient discrete log pseudo random generator, *Proc. of Crypto '98*, 304–317, 1998.

[Ri] R. Rivest, RC-6, *A proposal for the Advanced Encryption Standard*, 1998.

[SHA] U. S. Department of Commerce/ N. I. S. T, *Secure Hash Algorithm*, FIPS 180, April 1995.