International Journal of Security and Its Applications
Vol.9, No.11 (2015), pp.293-302
http://dx.doi.org/10.14257/ijsia.2015.9.11.27

# A Modular-Arithmetic-Based Encryption Scheme

†Odule, Tola John
Awodele O.

*Department of Mathematical Sciences Olabisi Onabanjo University*
*P.M.B. 2002, Ago-Iwoye, Ogun State. Nigeria.*
*tjodule@gmail.com*
*Department of Mathematics and Computer Science,*
*Babcock University*
*Ilishan-Remo, Ogun State. Nigeria.*
*awodeleo@babcock.edu.ng*

***Abstract***

*This paper considers a scenario in which a sender who holds a $k$-bit to $k$-bit trapdoor permutation $f$ wants to transmit a message $x$ to a receiver who holds the inverse permutation $f^{-1}$; with the condition that encryption should require just one computation of $f$, decryption should require just one computation of $f^{-1}$, the length of the enciphered text should be precisely $k$ and the length $n$ of the text $x$ that can be encrypted is close to $k$. Our scheme takes the encryption of $x$ to be $f(r_x)$, which is a simple probabilistic encoding of $x$. Assuming an ideal hash function and an arbitrary trapdoor permutation, we describe and prove secure a simple invertible enmesh scheme that is bit-optimal in that the length of the string $x$ that can be encrypted by $f(r_x)$ is almost $k$. Our scheme achieves semantic security, which implies chosen-cipher text security and non-malleability.*

***Keywords***: *Asymmetric encryption, provable security, trapdoor permutation, semantic security*

## 1. Introduction

Public-key encryption has been around for over thirty years. In its basic form, it is well understood: a public key allows for encryption, while an associated private (secret) key performs decryption. The complication lies in ensuring safe communication over an insecure channel in the presence of a malevolent eavesdropper, without the problem of key distribution and exchange, in a heterogeneous community of users.

In our setup we consider a sender who holds a $k$-bit to $k$-bit trapdoor permutation $f$ and wants to transmit a message $x$ to a receiver who holds the inverse permutation $f^{-1}$ We concentrate on the case which arises most often in cryptographic practice, where $n = |x|$ is at least a little smaller than $k$.

What practitioners want is the following encryption should require just one computation of $f$ decryption should require just one computation of $f^{-1}$ the length of the enciphered text should be precisely $k$ and the length $n$ of the text $x$ that can be encrypted is close to $k$ Since heuristic schemes achieving these conditions exist [1, 2], if

† Correspondence author

ISSN: 1738-9976 IJSIA

provable security is provided at the cost of violating any of these conditions, for instance two applications of $f$ to encrypt message length $n+k$ rather than $k$, practitioners will prefer the heuristic constructions. Thus to successfully impact practice one must provide provably-secure schemes which meet the above constraints.

The heuristic schemes invariably take the following form: one probabilistically, invertibly incorporates $x$ into a string $r_x$ and then takes the encryption of $x$ to be $f(r_x)$[1]. We call such a process an *invertible enmesh scheme*. We will take as our goal the construction of a provably *invertible enmesh schemes* which allows $n$ to be close to $k$.

Assuming an ideal hash function and an arbitrary trapdoor permutation, we describe and prove secure a simple *invertible enmesh scheme* that is bit-optimal in that the length of the string $x$ that can be encrypted by $f(r_x)$ is almost $k$. Our scheme achieves semantic security [3]. This notion is very strong, and in particular, implies ambitious goals like chosen-cipher text security and non-malleability [6] in the ideal-hash model which we assume.

### 1.1. The Basic Scheme

Recall $k$ is the security parameter, $f$ mapping $k$-bits to $k$-bits is the trapdoor permutation. Let $k_0$ be chosen such that the adversary's running time is significantly smaller than $2^{k_0}$ steps. We fix the length of the message to encrypt as $n = k - k_0$ bits (shorter messages can be suitably padded to this length). The scheme makes use of a *generator* $G: \{0,1\}^{k_0} \to \{0,1\}^n$ and a *hash function* $H: \{0,1\}^n \to \{0,1\}^{k_0}$. To encrypt $x \in \{0,1\}^n$ choose a random $k_0$ bit $r$ and set

$$\mathcal{E}^{G,H}(x) = f(x \oplus G(r)) \,\|\, r \oplus H(x \oplus G(r)))$$

Here $\|$ denotes concatenation. The decryption function $\mathcal{D}^{G,H}$ is defined in the obvious way, and the pair $(\mathcal{E}, \mathcal{D})$ constitutes what we call the *basic* scheme.

We prove security under the assumption that $G, H$ are *ideal*. This means $G$ is a random function of $\{0,1\}^{k_0} \to \{0,1\}^n$ and $H$ is a random function of $\{0,1\}^n \to \{0,1\}^{k_0}$. The formal statement of our result is in *Theorem 4.1*. It says that if $f$ is a trapdoor permutation and $G, H$ are ideal then the basic scheme achieves the notion of semantic security [3], appropriately adjusted to take account of the presence of $G, H$.

In practice, $G$ and $H$ are best derived from some standard cryptographic hash function. For example, they can be derived from the compression function of the Secure Hash Algorithm [7] following the methods described in [8].

### 1.2. Computational Efficiency of the Proposed Model

The function $f$ can be set to any candidate trapdoor permutation such as RSA [9] or modular squaring [10, 11]. In such a case the time for computing $G$ and $H$ is negligible compared to the time for computing $f$, $f^{-1}$. Thus complexity is discussed only in terms

[1]It is well-known that a naive enmesh like $r_x = x$ is no good; besides the usual deficiencies of any deterministic encryption, $f$ being a trapdoor permutation does not mean that $f(x)$ conceals all the interesting properties of $x$.Indeed it was exactly such considerations that helped inspire ideas like semantic security [3] and hardcore bits [4,5].

 

of $f$, $f^{-1}$ computations. In this light our basic encryption scheme requires just a single application of $f$ to encrypt, a single application of $f^{-1}$ to decrypt, and the length of the cipher text is $k$, as long as $k \geq n + k_0$. Our scheme requires a single application of $f$ to encrypt, a single application of $f^{-1}$ to decrypt, and the length of the cipher text is still $k$, as long as $k \geq n + k_0 + k_1$.

A concrete instantiation of our scheme, using RSA for $f$ and getting $G$, $H$ from the Secure Hash Algorithm [7], is given in Section 4.1.

### 1.3. The Ideal Hash Function Paradigm

As we indicated above, when proving security we take $G, H$ to be random, and when we want a concrete scheme. $G, H$ are instantiated by primitives derived from a cryptographic hash function. In this regard we are following the paradigm of [8] who argue that even though results which assume an ideal hash function do not provide provable security with respect to the standard model of computation, assuming an ideal hash function and doing proofs with respect to it provides much greater assurance benefit than purely *ad-hoc*, protocol design.

### 1.4. Assessment of the 'Exact Security' of our Scheme

We want our results to be meaningful for practice. In particular, this means we should be able to say meaningful things about the security of our schemes for specific values of the security parameter (e.g., $k = 512$). This demands not only that we avoid asymptotic and address security *exactly*, but also that we strive for security reductions which are as efficient as possible.[2]

Thus the theorem proving the security of our basic scheme, as in [12], quantifies the resources and success probability of a potential adversary let her run for time $t$ make $q_{gen}$ queries of $G$ and $q_{hash}$ queries of $H$, and suppose she could *break* the encryption with advantage $\varepsilon$. It then provides an algorithm $M$ and numbers $t', \varepsilon'$ such that $M$ inverts the underlying trapdoor permutation $f$ in time $t'$ with probability $\varepsilon'$. The strength of the result is in the values of $t', \varepsilon'$ which are specified as functions of $t$, $q_{gen}$, $q_{hash}$, $\varepsilon$, and the underlying scheme parameters $k$, $k_0$, $n$ $(k = k_0 + n)$. Now a user with some idea of the assumed strength of a particular $f$ such as RSA on $512$ bits can get an idea of the resources necessary to break our encryption scheme.

## 1. Notations and Conventions

### 2.1 Probabilistic algorithms

We hereby use the notation of [19]. If $A$ is a probabilistic algorithm then $A(x, y, \ldots)$ refers to the probability space which to the string $\sigma$ assigns the probability that $A$ on input $x, y, \ldots$ outputs $\sigma$. If $S$ is a probability space we denote its support, the set of

[2] Exact security is not new; previous works which address it explicitly include [13, 14, 15 16, 17, 18]. Moreover, although it is true that most theoretical works only provide asymptotic security guarantees of the form *the success probability of a polynomially bounded adversary is negligible* (everything measured as a function of the security parameter), the exact security can be derived from examination of the proof. (However, a lack of concern with the exactness means that in many cases the reductions are very inefficient, and the results are not useful for practice).

elements of positive probability, by $[S]$. When $S$ is a probability space, $x \leftarrow S$ denotes selecting a random sample from $S$. We use $x, y \leftarrow S$ as shorthand for $x \leftarrow S$; $y \leftarrow S$. For probability spaces $S, T, \ldots$, the notation $\Pr[x \leftarrow S; y \leftarrow T; \ldots, p(x, y, \ldots)]$ denotes the probability that the predicate $p(x, y, \ldots)$ is true after the ordered execution of the algorithms $x \leftarrow S$ $y \leftarrow T$, etc. PPT is short for *probabilistic-polynomial time*.

In evaluating the complexity of oracle machines we adopt the usual convention that all oracle queries receive their answer in unit time

### 2.2 Random Oracles

We will be discussing schemes which use functions $G, H$ chosen at random from appropriate spaces (the input and output lengths for $G$ and $H$ depend on parameters of the scheme). When stating definitions it is convenient to not have to worry about exactly what these spaces may be and just write $G, H \leftarrow \Omega$, the latter being defined as the set of all maps from the set $\{0,1\}^*$ of finite strings to the set $\{0,1\}^\infty$ of infinite strings. The notation should be interpreted as appropriate to the context--for example, if the scheme says $G$ maps $\{0,1\}^a$ to $\{0,1\}^b$ then we can interpret $G \leftarrow \Omega$ as meaning we choose $G$ from $\Omega$ at random, restrict the domain to $\{0,1\}^a$ and drop all but the first $b$ bits of output.

### 2.3 Trapdoor Permutations and their Security

Our encryption scheme requires a *trapdoor permutation generator*. This is a PPT algorithm $\mathcal{F}$ such that $\mathcal{F}(1^k)$ outputs a pair of deterministic algorithms $(f, f^{-1})$ specifying a permutation and its inverse on $\{0,1\}^k$.

We associate to $\mathcal{F}$ an evaluation time $T_{\mathcal{F}}(\cdot)$: for all $k$, all $(f, f^{-1}) \in [\mathcal{F}(1^k)]$ and all $w \in \{0,1\}^k$, the time to compute $f(w)$, given $f$ and $w$, is $T_{\mathcal{F}}(k)$. Note the evaluation time depends on the setting; for example on whether or not there is hardware available to compute $f$.

We will be interested in two attributes of a possibly non-uniform algorithm $M$ trying to invert $\mathcal{F}(1^k)$-distributed permutations, namely its running time and its success probability.

**Definition 1** Let $\mathcal{F}$ be a trapdoor permutation generator. We say that algorithm $M$ succeeds in $(t, \varepsilon)$-inverting $\mathcal{F}(1^k)$ if

$$\Pr[(f, f^{-1}) \leftarrow \mathcal{F}(1^k); \quad w \leftarrow \{0,1\}^k; \quad y \leftarrow f(w); \quad M(f, y) = w] \geq \varepsilon,$$

and, moreover, in the experiment above, $M$ runs in at most $t$ steps.

RSA [9] is a good candidate function as a secure trapdoor permutation.

 

## 2. Semantically Secure Encryption

We extend the definition of semantic security [3] to the random oracle model in a way which enables us to discuss exact security.

### 3.1 Encryption Schemes

An asymmetric encryption scheme is specified by a probabilistic generator, $\mathcal{G}$, and an associated plaintext-length function, $n$. On input $1^k$ the generator $\mathcal{G}$ outputs a pair of algorithms $(\mathcal{E}, \mathcal{D})$ the first of which is probabilistic. Each of these algorithms has oracle-access to two functions, one called $G$ and one called $H$. A user $i$ runs $G$ to get $(\mathcal{E}, \mathcal{D})$ and makes the former public while keeping the latter secret. To encrypt message $x \in \{0, 1\}^{n(k)}$ using functions $G, H$ anyone can compute $y = \mathcal{E}^{G,H}(x)$ and send it to $i$. To decrypt cipher text $y$ user $i$ computes $x = \mathcal{D}^{G,H}(y)$. We require $\mathcal{D}^{G,H}(y) = x$ for all $y \in \left[\mathcal{E}^{G,H}(x)\right]$. We further demand that $\mathcal{D}^{G,H}(y) = *$ if there is no $x$ such that $y \in \left[\mathcal{E}^{G,H}(x)\right]$.

An adversary is a possibly non-uniform algorithm $A$ with access to oracles $G, H$. We assume without loss of generality (w.l.o.g) that an adversary makes no particular $G$-query more than once and no particular $H$-query more than once. For simplicity we assume that the number of $G$-queries and $H$-queries that an adversary makes don't depend on its coin tosses but only, say, on the length of its input.

### 3.2 Semantic Security

The following definition will be used to discuss *exact* security. It captures the notion of semantic security [3] appropriately lifted to take into account the presence of $G, H$.

We consider an adversary who runs in two stages. In the **find**-stage it is given an encryption algorithm $\mathcal{E}$ and outputs a pair $x_0, x_1$ of messages. It also outputs a string $c$ which could record, for example, its history and its inputs. Now we pick at random either $x_0$ or $x_1$, the choice made according to a bit $b$, and encrypt it under $\mathcal{E}$ to get $y$. In the **guess**-stage we provide $A$ the output $x_0, x_1, c$ of the previous stage, and $y$, and we ask it to guess $b$. We assume w.l.o.g that $\mathcal{E}$ is included in $c$ so that we don't need to explicitly provide it again. Since even the algorithm which always outputs a fixed bit will be right half of the time, we measure how well $A$ is doing by $1/2$ less than the fraction of time that $A$ correctly predicts $b$. We call twice this quantity the advantage which $A$ has in predicting $b$. Multiplying by two makes the advantage fall in the range $[0, 1]$ (0 for a worthless prediction and 1 for an always correct one), instead of $[0, 0.5]$.

**Definition 2** Let $\mathcal{G}$ be a generator for an encryption scheme having plaintext-length function $n(\cdot)$. An adversary $A$ is said to succeed in $(t, q_{gen}, q_{hash}, \varepsilon)$-breaking $\mathcal{G}(1^k)$

if $\varepsilon \leq 2 \cdot \Pr\left[\begin{array}{c}(\mathcal{E}, \mathcal{D}) \leftarrow \mathcal{G}(1^k);\ G, H \leftarrow \Omega,\ (x_0, x_1, c) \leftarrow A^{G,H}(\mathcal{E}, \textbf{find}) \\ b \leftarrow \{0, 1\};\quad y \leftarrow \mathcal{E}^{G,H}(x_b);\quad A^{G,H}(y, x_0, x_1, c) = b\end{array}\right] - 1,$

And, moreover, in the experiment above, $A$ runs for at most $t$ steps, makes at most $q_{gen}$ queries to $G$ and makes at most $q_{hash}$ queries to $H$.

Note that $t$ is the total running time, that is, the sum of the times in the two stages. Similarly, $q_{gen}$, $q_{hash}$ are the total number of $G$ and $H$ queries, respectively.

## 3. The Basic Encryption Scheme

Let $\mathcal{F}$ be a trapdoor permutation generator and $k_0(\cdot)$ a positive integer valued function such that $k_0(k) < k$ for all $k \geq 1$. The basic scheme $\mathcal{G}$ with parameters $\mathcal{F}$ and $k_0(\cdot)$ has an associated plaintext-length function of $n(k) = k - k_0(k)$. On input $1^k$, the generator $\mathcal{G}$ runs $\mathcal{F}\left(1^{\mathrm{k}}\right)$ to obtain $(f, f^{-1})$. Then it outputs the pair of algorithms $(\mathcal{E}, \mathcal{D})$ determined as follows:

1. On input $x$ of length $n = n(k)$, algorithm $\mathcal{E}$ selects a random $r$ of length $k_0 = k_0(k)$ It sets $s = x \oplus G(r)$ and $t = r \oplus H(s)$. It sets $w = s \| t$ and returns $y = f(w)$.
2. On input $y$ of length $k$, algorithm $\mathcal{D}$ computes $w = f^{-1}(y)$. Then it sets $s$ to the first $n$ bits of $w$ and $t$ to the last $k_0$-bits of $w$. It sets $r = t \oplus H(s)$, and returns the string $x = s \oplus G(r)$.

The oracles $G$ and $H$ which $\mathcal{E}$ and $\mathcal{D}$ reference above have input-output lengths of $G: \{0,1\}^{k_0} \rightarrow \{0,1\}^n$ and $H: \{0,1\}^n \rightarrow \{0,1\}^{k_0}$. We use the encoding of $f$ as the encoding of $\mathcal{E}$ and the encoding of $f^{-1}$ as the encoding of $\mathcal{D}$.

The intuition behind the semantic security of this scheme is as follows. We wish to guarantee that the adversary, given a point $y$ in the range of $f$, must recover the complete preimage $w = r_x$ of $y$ if she is to say anything meaningful about $x$ itself. Well, if the adversary does not recover all of the first $n$ bits of the preimage, $s$, then she will have no idea about the value $H(s)$ which is its hash; a failure to know anything about $H(s)$ implies a failure to know anything about $r = H(s) \oplus t$ ,where $t$ is the last $k_0$ bits of $w$, and therefore $G(r)$ and therefore $x = G(r) \oplus s$ itself. Now, assuming the adversary does recover $s$, a failure to completely recover $t$ will again mean that the adversary fails to completely recover $r$, and, in the lack of complete knowledge about $r$, $x \oplus G(r)$ is uniformly distributed and so again the adversary can know nothing about $x$.

Yet the above discussion masks some subtleties and a formal proof of security is more complex than it might appear. This is particularly the case when one is interested, as we are here, in achieving the best possible exact security.

The following theorem says that if there is an adversary $A$ who is able to break the encryption scheme with some success probability, then there is an algorithm $M$ which can invert the underlying trapdoor permutation with comparable success probability and in comparable time. This implies that if the trapdoor permutations can't be inverted in reasonable time (which is the implicit assumption) then our scheme is secure. But the theorem says more; it specifies exactly how the resources and success of $M$ relate to those of $A$ and to the underlying scheme parameters $k, n, k_0$ $(k = n + k_0)$.

 

The inverting algorithm $M$ can by obtained from $A$ in a *uniform* way; the theorem says there is a *universal* oracle machine $U$ such that $M$ can be implemented by $U$ with oracle access to $A$. It is important for practice that the *description* of $U$ is *small*; this is not made explicit in the theorem but is clear from the proof. The constant $\lambda$ depends only on details of the underlying model of computation. We write $n, k_0$ for $n(k), k_0(k)$, respectively, when, as below, $k$ is understood.

**Theorem 4.0**Let $G$ be the basic encryption scheme with parameters $\mathcal{F}$, $k_0$ and let $n$ be the associated plaintext length. Then there exists an oracle machine $U$ and a constant $\lambda$ such that for each integer $k$ the following is true. Suppose $A$ succeeds in $(t, q_{gen}, q_{hash}, \varepsilon)$-breaking $G\left(1^k\right)$. Then $M = U^A$ succeeds in $(t', \varepsilon')$-inverting $\mathcal{F}\left(1^k\right)$ where

$$t' = t + q_{gen} \cdot q_{hash} \cdot \left(T_{\mathcal{F}}(k) + \lambda k\right)$$
$$\varepsilon' = \varepsilon \cdot \left(1 - q_{gen} 2^{-k_0} - q_{hash} 2^{-n}\right) - q_{gen} 2^{-k+1}.$$

We omit the proof of this *Theorem* for the sake of brevity.

For reasonable values of $k$ (e.g., $k \geq 512$) it will be the case that $k > n >> k_0$. Thus for reasonable values of $q_{gen}, q_{hash}$ we'll have $\varepsilon' \approx \varepsilon \cdot \left(1 - q_{gen} 2^{-k_0}\right)$. Thus the success probability $\varepsilon'$ achieved here is good in the sense that it is only slightly less than $\varepsilon$, and close to optimal. Note also that the expression for $\varepsilon'$ indicates that $A$ will do best by favouring $G$-oracle queries over $H$-oracle queries.

The dominant factor in the time $t'$ taken by the inverting algorithm to compute $f^{-1}(y)$ is the time to do $q_{gen} \cdot q_{hash}$ computations of the underlying $f$. An interesting open question is to find a scheme under which the number of computation of $f$ is linear in $q_{gen} + q_{hash}$ while retaining a value of $\varepsilon'$ similar to ours.

### 4.1 A Prototype Model of our Scheme

We provide here a concrete implementation of our encryption scheme, omitting only certain minor details. We use RSA as the trapdoor permutation and construct the functions $G, H$ out of the revised NIST Secure Hash Algorithm [7], although other hash algorithms such as MD5 [20] would do as well.

Let $f$ be the RSA function [9], so $f(x) = x^e \bmod N$ is specified by $(e, N)$ where $N$ is the $k$-bit product of two large primes and $(e, \varphi(N)) = 1$. We demand $k \geq 512$ bits, larger values are recommended.

Our scheme will allow the encryption of any string $msg$ whose length is at most $k - 320$ bits; thus the minimal permitted security parameter allows 192 bits (e.g., three 192-bit keys to be encrypted). Let $D = \{1 \leq i < N; \ \gcd(i, N) = 1\} \subseteq \{0, 1\}^k$ D be the set of valid domain points for $f$.

Our probabilistic encryption scheme depends on the message $msg$ to encrypt, an arbitrary-length string *rand_coins,* the security parameter $k$, the function $f$ and a predicate $IN\ D(x)$ which should return t**rue** if and only if $x \in D$. Our scheme further uses a 32-bit string $key\_data$, whose use we do not specify here, and a string $desc$

which provides a complete description of the function $f$; that is, it says "This is RSA using $N$ and $e$" encoded according to conventions not specified here.

We denote by $\mathrm{SHA}_\sigma(x)$ the 160-bit result of SHA (Secure Hash Algorithm applied to $x$, except that the 160-bit *starting value* in the algorithm description is taken to be $ABCDE = \sigma$.. Let $\mathrm{SHA}_\sigma^\ell(x)$ denote the first $\ell$-bits of $\mathrm{SHA}_\sigma(x)$. Fix the notation $\langle i \rangle$ for $i$ encoded as a binary 32-bit word.

We define the function $H_\sigma^\ell(x)$ for string $x$, number $\ell$ and 160-bit $\sigma$ to be the $\ell$-bit prefix of $\mathrm{SHA}_\sigma^{80}(\langle 0 \rangle \cdot x) \parallel \mathrm{SHA}_\sigma^{80}(\langle 1 \rangle \cdot x) \parallel \mathrm{SHA}_\sigma^{80}(\langle 2 \rangle \cdot x) \parallel \cdots$

Let $K_0$ be a fixed, randomly-chosen 160-bit string, which we do not specify here.

Our scheme is depicted in Figure 1. Basically, we augment the string $msg$ which we want to encrypt by tacking on a word to indicate its length; including $k_1 = 128$ bits of redundancy; incorporating a 32-bit field $key\_data$ whose use we do not specify, and adding enough additional padding to fill out the length of the string we have made to $k - 128$ bits. The resulting string $x$ now plays the same role as the $x$ of our basic scheme, and a separate 128-bit $r$ is then used to encrypt it.

ENCRYPT ( *msg* ,*rand_coins*)

$\sigma = \mathrm{SHA}_{K_0}(desc)$;
$\sigma_1 = \mathrm{SHA}_\sigma(\langle 1 \rangle)$;
$\sigma_2 = \mathrm{SHA}_\sigma(\langle 2 \rangle)$;
$\sigma_3 = \mathrm{SHA}_\sigma(\langle 3 \rangle)$;
$i \leftarrow 0$;
**repeat**
  $r \leftarrow H_{\sigma_1}^{128}(\langle i \rangle \parallel rand\_coins)$;
  $x \leftarrow key\_data \parallel \langle |msg| \rangle \parallel 0^{128} \parallel 0^{k-320-|msg|} \parallel msg$;
  $\bar{x} \leftarrow x \oplus H_{\sigma_2}^{|x|}(r)$;
  $\bar{r} \leftarrow r \oplus H_{\sigma_3}^{128}(\bar{x})$;
  $r_x = \bar{x} \parallel \bar{r}$;
  $i \leftarrow i + 1$;
  **until** in $\mathrm{D}(r_x)$;
  **return** $f(r_x)$;

**Figure 1. A Sample Implementation of our Encryption Scheme.**

## 4. Conclusion

We comment that in the concrete scheme shown in Figure 1 we have elected to make our generator and hash function sensitive both to our scheme itself, via $K_0$ and to the particular function $f$, via $desc$. Such *key separation* is a generally-useful heuristic to help ensure that when the same key is used in multiple, separately-secure, algorithms that

 

the internals of these algorithms do not interact in such a way as to jointly compromise security. The use of *key variants* $\sigma_1$, $\sigma_2$ and $\sigma_3$ is motivated similarly. Our choice to only use half the bits of SHA has to do with a general deficiency in the use of SHA-like hash functions to implement random oracles.

## References

[1] RSA Data Security, Inc., "PKCS #1: RSA Encryption Standard," June 1991.
[2] M. Bellare and P. Rogaway, "Random oracles are practical: a paradigm for designing efficient protocols." Proceedings of the First Annual Conference on Computer and Communications Security, ACM. 1993.
[3] S. Goldwasser and S. Micali, "Probabilistic Encryption," Journal of Computer and System Sciences **28**, 270-299, April 1984.
[4] M. Blum and S. Micali, "How to generate cryptographically strong sequences of pseudo- random bits," SIAM Journal on Computing **13** (4), 850-864, November 1984.
[5] A. Yao, "Theory and applications of trapdoor functions," Proceedings of the 23rd Symposium on Foundations of Computer Science, IEEE, 1982.
[6] D. Dolev, C. Dwork and M. Naor, "Non-malleable cryptography," Proceedings of the 23rd Annual Symposium on Theory of Computing, ACM,1991.
[7] National Institute of Standards, FIPS Publication 180, "Secure Hash Standard," 1993.
[8] D. Johnson, A. Lee, W. Martin, S. Matyas and J. Wilkins, "Hybrid key distribution scheme giving key record recovery," IBM Technical Disclosure Bulletin, **37** (2A), 5-16,February 1994.
[9] R. Rivest, A. Shamir and L. Adleman, "A method for obtaining digital signatures and public key cryptosystems," CACM 21 (1978).
[10] M. Rabin, "Digitalized signatures and public-key functions as intractable as factorization," MIT Laboratory for Computer Science TR 212, January 1979.
[11] . Blum, M. Blum and M. Shub, "A Simple Unpredictable Pseudo-Random Number Generator," SIAM Journal on Computing **15**(2), 364-383, May 1986.
[12] Odule, T.J. (2007): Incremental Cryptography and Security of Public Hash Functions. Journal of Nigerian Association of Mathematical Physics, vol. 11 pp. 467-474
[13] O. Goldreich and L. Levin, "A hard predicate for all one-way functions," Proceedings of the 21st Annual Symposium on Theory of Computing, ACM,1989.
[14] R. Impagliazzo, L. Levin and M. Luby, "Pseudo-random generation from one-way functions," Proceedings of the 21st Annual Symposium on Theory of Computing, ACM, 1989.
[15] C. Schnorr, "Efficient identification and signatures for smart cards," Advances in Cryptology - Crypto 89 Proceedings, Lecture Notes in Computer Science Vol. 435, G. Brassard ed., Springer-Verlag,1989.
[16] . Leighton and S. Micali, "Provably fast and secure digital signature algorithms based on secure hash functions," Manuscript, March _____
[17] S. Evens, O. Goldreich and S. Micali, "On-line/Off line digital signatures," Manuscript, Preliminary version in Advances in Cryptology - Crypto 89 Proceedings, Lecture Notes in Computer Science Vol. 435, G. Brassard ed., Springer-Verlag, 1989.
[18] M. Bellare, J. Kilian and P. Rogaway, "On the security of cipher-block chaining," Advances in Cryptology - Crypto 94 Proceedings, Lecture Notes in Computer Science Vol. 839, Y. Desmedt ed., Springer-Verlag, 1994.
[19] . Goldwasser, S. Micali and R. Rivest, "A digital signature scheme secure against adaptive chosen-message attacks," SIAM Journal of Computing,**17**(2), 281-308, April 1988.
[20] . Rivest, "The MD5 message-digest algorithm," IETF Network Working Group, RFC 1321, April 1992.

 