Design of a Synchronous Stream Cipher from Hash Functions

Angelo P. E. Rosiello

Summary
We consider a simple and secure way to realize a synchronous stream cipher from iterated hash functions.
It is similar to the OFB mode where the underlying block cipher algorithm is replaced with the keyed hash function, adopting the secret suffix method[20].
We analyzed the key, the keystream and the necessary properties to assume from the underlying hash function for the stream cipher to be considered secure. Motivated by our analysis we conjecture that the most efficient way to break the proposed stream cipher is to break the hash function or through exhaustive search for the keyspace *K* of *k* bits, that requires O(2^k) operations.

Key words:
stream cipher, key, keystream, one-time pad cryptosystem, hash function, keyed hash function.

1. Security Requirements of the Stream Cipher

The algorithm should have a flat keyspace allowing any random bit string to be a possible key. The algorithm should make easier the key-management for software implementations. The typed password should not become directly the key, else the actual keyspace is limited to keys constructed with the 95 characters of printable ASCII.
The algorithm should be easily modifiable satisfying minimum or maximum requirements. Moreover, according to basic engineering software theories, the algorithm does not have to bind developers with static use of pre-defined logical block functions, but it is important to let wide alternatives during the implementation of the software[13].
The algorithm should be simple to code, otherwise programmers could make implementation mistakes if the structure is too complicated.
Nowadays, encrypting information has become a "must", which means that a good crypto algorithm must give to the community the possibility to manage safe data.
Practical applications pertain to:

- **Bulk Encryption**: data files or a continuous data stream (e.g. important information saved on hardisks such as databases or any kind of secret document);
- **Data Transmission**: a lot of communication mediums need a secure way to crypt exchanged information (e.g. Internet packets, wireless connections, radio signals, etc.);
- **Small Encryption**: banks and commercial companies need secure encryption methodologies to interact with customers by small encryption technologies.

Definitely, a good algorithm should be suitable for lots of disparate situations.
The paradigm is to approach, as much as possible, to the ideal features constituting the one-time pad cryptosystem. The one-time pad cipher is unconditionally secure as Shannon proved in his seminal paper[9]. According to the one-time pad cipher design, the length of the key must be at least as long as the plaintext and it must be completely random. The one-time pad can offer the maximum security degree but it is hard and really expensive to be realized, in fact the secure distribution of the required keying material would pose an enormous logistical problem if the one-time pad were used on a large scale.
Similarly to all the existing stream ciphers, our algorithm presents two main periods:

1. During the first period the *key* and the *keystream* are generated using a hash function with scientific criteria.
2. The second period will apply to final computational step, producing the ciphertext.

Definitions:

Plaintext message *m* is a sequence of bits $m_0 m_1 \dots m_{n-1}$;

Ciphertext message *c* is a sequence of $c_0 c_1 \dots c_{n-1}$;

Keystream is a pseudo-random sequence of bits $k_0 k_1 \dots k_{n-1}$;

$c_i = m_i \oplus k_i$

for $0 \le i \le n-1$ where $\oplus$ denotes bitwise exclusive-or.

2. Design of the Stream Cipher

In this chapter the design of the algorithm will be described in its theoretical essence, hinting sometimes at his probable practical implementation.

Manuscript received August 5, 2007
Manuscript revised August 20, 2007

2.1 Generation of the Key

During this phase the key is generated applying HMAC[11] to the input password.

2.1.1 The Function HMAC

HMAC is an adaptation of NMAC[11] that uses directly the iterated hash function (with its defined and fixed IV) as the basic black-box to build the MAC. Denoted with *F* the (iterated and keyless) hash function initialized with its own IV value, with *x* any input of arbitrary length and with *k* the key:

$$HMAC_k(x) = F(k \parallel pad_1 \parallel F(k \parallel pad_2 \parallel k))$$
$$= HMAC_k(k)$$

where pad_1 and pad_2 are distinct strings of sufficient length to pad *k* out to a full block for the compression function of the hash function.

2.1.2 Key of the Stream Cipher

The key is the result of the function HMAC to the input password *k*:

$key = F(k \parallel pad_1 \parallel F(k \parallel pad_2 \parallel k) = HMAC_k(k)$ wher e F is the hash function with properties required in §4.2.

2.1 Generation of the Keystream

During this phase the whole keystream is generated constantly depending on the key and its past. The process is realized applying two similar functions named *p* and *q* and defined as follows:

Let $p : \{0,1\}^+ \rightarrow \{0,1\}^+$ be a functions such that,

$$p(x) = LSB_n(x), |x|/2 \le n \le |x|$$

Let $q : \{0,1\}^+ \rightarrow \{0,1\}^*$ be a functions such that,

$$q(x) = LSB_m(x), 0 \le m \le |x|$$

Once that *n* is fixed it must be the same for all the generation of the keystream while *m* can assume different values for a mono-dimensional matrix $M = (m_1, m_2, \ldots, m_i)$ so that $q(x_i) = LSB_{m_i}(x_i), \forall i \ge 1$.

The keystream is generated as follows:

$$y(1) = f\_hash(key)$$
$$y(2) = f\_hash(p(y1) \parallel key)$$
$$y(3) = f\_hash(p(y(2)) \parallel q(y(1)) \parallel key)$$
$$\vdots$$
$$y(n) = f\_hash(p(y(n-1)) \parallel q(y(n-2)) \parallel \ldots \parallel$$
$$\parallel q(y(n-(n-1))) \parallel key)$$
$$keystream = y(1) \parallel y(2) \parallel \ldots \parallel y(n)$$

Denoted with *l* the length of the output of the hash function, the function *p* guarantees a good level of security for the stream cipher and at least $2^{l/2}$ different possible outputs of the hash function. However, it is not the optimal security choice considering only the function *p* to realize the cipher without a significant contribution from the function *q*. It is strongly suggested to set $n = 2^{l/2}$ and $m_1 + m_2 + \ldots + m_i = l/2$ (more bits could compromise the performance without any security advantage) possibly not restricting only to m_1, to achieve a better security and to let the range of the hash function, theoretically (excluding collisions), equal the co-domain (i.e. 2^l).

The exact value of each m_i should be set in relation to the context. For example, in order to minimize the collision effects of the hash function (if it is weakly collision resistant, even if it should not be), one should take some bits from different positions of the past keystream, in fact, under certain conditions, a collision could compromise the security of the stream cipher.

3. Properties of the Underlying Hash Function

The hash function *f* used to design the stream cipher is the cryptographic core of the stream cipher itself. It must belong to the MD4-family (i.e. iterated hash functions) and satisfy some important requirements.

Preimage resistance. Given a hashed value *h*, it should be computationally infeasible to find an input *x* such that *h=f(x)*.

Partial-preimage resistance. It should be as difficult to recover any substring as to recover the entire input.

Moreover, even if part of the input is known, it should be difficult to find the remainder (e.g., if t input bits remain unknown, it should take on average 2^{t-1} hash operations to find these bits).

Collision resistance. It should be computationally infeasible to find two inputs *x,y* with $x \neq y$ such that $f(x) = f(y)$.

Mixing-Transformation. On any input *x*, the output hashed value $h = f(x)$ should be computationally indistinguishable from a uniform binary string. Here, the computational indistinguishability follows Definition 4.15 (in §4.7) in [8].

Theorem 1: *Let f be the underlying iterated hash function of the proposed stream cipher. The following properties of f are necessary condition for the security of the stream cipher: preimage resistance, partial-preimage resistance, collision resistance and mixing-transformation.*

The theorem states that all above properties of the hash function are necessary for the security of the stream cipher, thus, if one of them should not be respected the security would be compromised (at least theoretically).

Proof of Theorem 1:

1. **Preimage resistance.** Let *f* be the underlying hash function of the *secure* stream cipher and *h* the hashed value. Suppose, for absurd, that *f* is not preimage resistant, therefore, it is feasible to find the input *x* such that *h = f(x)*. In a known-plaintext attack the adversary can recover the key, that is in the keystream preimage with probability 1 and minimum computational power, contradicting the above mentioned security of the stream cipher.
2. **Partial-preimage resistance.** Let *f* be the underlying hash function of the *secure* stream cipher and *h* the hashed value. Suppose, for absurd, that *f* is not partial-preimage resistant, therefore, it is feasible to find the whole input *x*, such that *h = f(x)*, if *x* is partially known. In a known-plaintext attack the adversary, that knows part of the input *x* (except the key) used to generate the keystream, can smoothly recover the remaining input containing the key, contradicting the above mentioned security of the stream cipher.
3. **Collision resistance.** Let *f* be the underlying hash function of the *secure* stream cipher. Suppose, for absurd, that *f* is not collision resistant, therefore, it is recurrent to find two inputs *x,y* with $x \neq y$ such that $f(x) = f(y)$. During the generation of the keystream of the stream cipher's, if *n* has the same value of the length of the output of the hash function and *M* is the banal matrix, it can happen that a very short cycle of bits build the keystream, as shown below:

   $y(1) = f(key)$

   $y(2) = f(p(y(1)) \,||\, key)$

   Suppose *p(x)=x* and a collision happened:

   $y(2) = y(1)$

   Suppose that:

   $M = (0,0,\ldots,0) \Rightarrow \forall x q(x) = \varepsilon$

   Where ε denotes the empty string, then:

   $y(3) = f(p(y(2)) \,||\, q(y(1)) \,||\, key)$

   $= f(p(y(1)) \,||\, key) = y(2) = y(1)$

   $\vdots$

   $y(1) = y(2) = y(3) = \ldots = y(n)$

   In a known-plaintext attack, the adversary can reproduce the whole keystream, that is the same cyclical set of few bits, recovering the plaintext, contradicting the stream cipher's above mentioned security.
4. **Mixing-Transformation.** Let *f* be the underlying hash function of the *secure* stream cipher. Suppose, for absurd, that *f* does not realize a mixing-transformation, therefore, the output hashed value *h = f(x)* is polinomially distinguishable from a truly random binary string. The attacker may be able to predict the input/output with non-negligible advantage $\varepsilon > 0$, recovering the keystream, contradicting the above mentioned security of the stream cipher.

At last, hash functions with less than 160 bits of output should not be considered.

4. Analysis of the Key

The key of the stream cipher is obtained using the function HMAC. The security of HMAC is based on the security of the NMAC construction, which is stated from Theorem 4.1 and its proof in [11]. HMAC is a particular case of NMAC where k_1 and k_2 derive using the compression function *f* of the hash function *h* and they cannot be distinguished by the attacker from truly random keys. De facto, a weak pseudo-randomness of *f* is required since the attacker that is trying to find out the dependencies of k_1 and k_2 cannot see directly the output of the pseudo-random function on any input.

In summary, as claimed by Bellare, Canetti and Krawczyk in [11]:

"Attacks that works on HMAC and not on NMAC are possible, in principle. However, such an attack would reveal significant weaknesses of the pseudorandom properties of the underlying hash function, contradicting in a strong sense the usual assumptions on these functions".

5. Analysis of the Keystream: Keyed Hash Function

Last years several ways were proposed to key with *k* bits an unkeyed iterated hash function *h* of *n* output bits, mainly to realize a secure MAC. The most important are described below, evidencing their known weaknesses.

- **The Secret Prefix Method.** This method consists of prepending a secret key *K* to the message *x* before the hashing operation: $MAC(x) = h(k \| x)$. This MAC is insecure, in fact, a single text-MAC pair contains information essentially equivalent to the secret key, independent of its size[10].
- **The Secret Suffix Method.** This method consists of appending a secret key *K* to the message *x* before the hashing operation: $MAC(x) = h(x \| k)$. Adopting this way, an off-line collision attack on the hash function may be used to obtain an internal collision; therefore by a birthday attack finding a collision requires $O(2^{n/2})$ off-line operations. Besides, given one known text-MAC pair, it is possible to performe an existential MAC forgery if a second preimage attack on the hash function is feasible. If *t* text-MAC pairs are known, finding a MAC second preimage requires $2^{n/t}$ trials; if the length of the message is not appended *t* denotes the number of blocks rather than the number of messages[10]. In order to recover the key are needed $O(2^k)$ operations and known MAC-text pairs for the verification.
- **The Envelope Method.** This method combines the prefix and the suffix methods. It consist of prepending a secret key k_1 and appending a secret key k_2 to the message *x* before the hashing operation: $MAC(x) = h(k_1 \| x \| k_2)$. As shown in [10], this method is also subject to the forgery and it is possible to apply a divide and conquer key recovery attack on k_1 and k_2 so that with $2^{n/2}$ known text-MAC pairs, one can recover the key with $2^{k_1} + 2^{k_2}$ instead of $2^{k_1+k_2}$ trials.
- **MDx-MAC/NMAC/HMAC.** These methods are dedicated constructions to build a secure MAC from an unkeyed hash function and to avoid all the known attacks.

In the proposed stream cipher, the generation of the keystream is done by keying the iterated hash function with the secret suffix method, because it is evidently more efficient than the dedicated MAC constructions(i.e. MDx-MAC, NMAC, HMAC) but, in this context secure, too.

The secret suffix method is shown to be weak to the *forgery* if a second preimage attack on the hash function is feasible[10], but the hash function for the proposed cipher has to be collision resistant which implies the second preimage resistance property.

In a known-plaintext attack scenario, where $\lceil k/n \rceil$ text-MAC pairs are known, the adversary should compute $O(2^k)$ off-line operations to recover the key[10].

Finally, in a known-key attack scenario, where *t* bits of the key remain unknown and $\lceil k/n \rceil$ text-MAC pairs are known, it should take on average 2^{t-1} hash operations to find these bits because of the partial-preimage resistance of the hash function.

6. Conclusions

An important aspect of this work is to consider the hash function as a black-box. In fact, the hash function can be seen as a module that can be replaced in case serious weaknesses are found in the hash function or when new more secure or efficient hash function are designed.

We conjecture that the most efficient way to break the proposed stream cipher is to break the underlying hash function or through exhaustive search for the keyspace *K* of *k* bits, that requires $O(2^k)$ operations.

In fact, it is true that the pseudo-randomness of the keystream is unconditionally secure only under the random oracle model but a ROM-based security proof suggests that for a real world encryption scheme which uses real world hash functions rather than ROs, the most vulnerable point to mount an attack is the hash function used in the scheme[8]. Since breaking suitable real world iterated hash functions such as RIPEMD-160[2] or SHA-1[1] is considered a hard problem, breaking the stream cipher shold be, too.

The complexity of the algorithm is embedded in the one-way hash function.

References

[1] Secure Hash Standard. http://www.itl.nist.gov/fipspubs/fip180-1.htm, 1995 April 17

[2] Hans Dobbertin, Antoon Bosselaers, and Bart Preneel. RIPEMD-160. http://www.esat.kuleuven.ac.be/~bosselae/ripemd160.html

[3] M.J.B. Robshaw. Stream Ciphers. RSA Laboratories Technical Report TR-701 Version 2.0 July 25, 1995

[4] RSA Laboratories. Answers to Frequently Asked Questions About Today's Cryptography Revision 2.0, RSA Data Security Inc., 5 Oct 1993.

[5] E. MendelsonThe language of first order logic. Cambridge University Press, 1993.

[6] Codes and Cryptography. Dominic Welsh. Codes and Cryptography. Oxford University Press, 1988.

[7] Thomas H. Cormen, Charles E: Leiserson, and Ronald L. Rivest. Introduction to Algorithms. The MIT Press, Cambridge, MA, 1990.

[8] Wenbo Mao. Modern Cryptography Theory and Practice. Prentice Hall PTR, 2004.

[9] C.E. Shannon. Communication theory of secrecy systems. Bell System Technical Journal, 28:657-715, 1949.

[10] Bart Preneel, Paul C. van Oorscot. MDx-MAC and Building Fast MACs from Hash Functions. Springer-Verlag LNCS, August 1995.

[11] M. Bellare, R. Canetti, and H. Krawczyk. Keying Hash Functions for Message Authentication. Advances in Cryptology - Crypto 96 Proceedings, Lecture Notes in Computer Science Vol. 1109, N. Koblitz ed, Springer-Verlag, 1996.

[12] Stephen Bernstein, Ruth Bernstein, Schaums. Schaum's Outline of Elements of Statistics I: Descriptive Statistics and Probability. McGraw-Hill, 1998.

[13] Barbara Liskov, John V. Guttag. Abstraction and Specification in Program Development. McGraw Hill Text, December 1986.

[14] Paolo Baldi. Introduzione alla Probabilità con elementi di Statistica. McGraw-Hill, 2003.

[15] Clifford A. Shaffer. A Practical Introduction to Data Structures and Algorithm Analysis. Prentice Hall, 1998.

[16] B. Schneier. Applied Cryptography. John Wiley \& Sons, New York, 1994

[17] Frederick P.Brooks. The Mythical Man-Month: Essays on Software Engineering. Addison-Wesley, Reading, MA, 1975.

[18] A. Menezes, P. van Oorschot, and S. Vanstone. Handbook of Applied Cryptography. CRC Press, 1996.

[19] R.J. Enbody and H.C. Du. Dynamic Hashing schemes. Computing surveys, 1988.

[20] G.Tsudik. Message authentication with one-way hash functions. ACM Computer Communications Review, Vol. 22, No. 5, 1992, pp. 29-38.

[21] P. Gutmann, personal communication, 1993.

Angelo P. E. Rosiello received the B.S. and M.S. degrees in Computer Science Engineering *cum laude* from "Politecnico di Milano" in 2004 and 2006, respectively. At the moment, Angelo works for Accenture in the Security Service Line. He also collaborates with Prof. Christopher Kruegel and Prof. Engin Kirda (Technical University of Vienna) in the ICT security field.