ARC: A Synchronous Stream Cipher from Hash Functions

Angelo P. E. Rosiello
Roberto Carrozzo

22nd December 2005

Contents

Abstract

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.

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

1 Basic Notions

Stream Cipher Stream ciphers are an important class of encryption algorithms. They encrypt individual characters (usually binary digits) of a plaintext message one at a time, using an encryption transformation which varies with time.

Keystream A stream cipher generates what is called a keystream (a sequence of bits used as a key) usually to be XOR-ed with the plaintext or the ciphertext.

Synchronous Stream Cipher A synchronous stream cipher is one in which the keystream is generated independently of the plaintext message and of the ciphertext.

One-Time Pad A one-time pad is a cryptosystem that uses a string of bits that is generated completely at random. The keystream is the same length as the plaintext message and the random string is combined using bitwise XOR with the plaintext to produce the ciphertext.

Hash Function A hash function (in the unrestricted sense) is a function h which has, as a minimum, the following two properties[18]:

1. compression - h maps an input x of arbitrary finite bitlength, to an output $h(x)$ of fixed bitlength n.
2. ease of computation - given h and an input x, $h(x)$ is easy to compute.

Keyed Hash Function A keyed hash function is a hash function with a secondary input, the secret key K.

MAC A message authentication code (MAC) is an authentication tag derived by applying an authentication scheme, together with a secret key, to a message.

1.1 Algorithm Requirements

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[1].

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, 17].

The algorithm should be simple to code, otherwise programmers could make implementation mistakes if the structure is too complicated[13].

1.2 Areas of Application

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.

[1]Non ASCII codifications can be considered, too.

2 Vision

The paradigm is to approach, as much as possible, to the ideal features constituting the one-time pad cryptosystem.

2.1 The perfect algorithm

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[3, 9].

The Shannon Theorem shows that the one-time pad is secure but does not express a paradigm for the perfect cryptosystem. We decided to describe it with the language of first order logic[5].

$\mathcal{D} = \mathcal{N}$: Domain
$x1, x2, x3, x4$: messages
$\oplus$: operator[2]
$|x| : |\emptyset| = 0 \wedge |x|.'y' = |x| + 1$
$crypt(x1, x2)$: predicate; $crypt(x1, x2) \leftrightarrow x1 \oplus x2$
$random(x)$: predicate[3]

A1. $x1 = x1$
A2. $x1 = x1 \wedge x2 = x3 \rightarrow x1 = x3$
A3. $x1 = x2 \wedge x2 = x1$
A4. $x1 \oplus x2 = x2 \oplus x1$
A5. $x1 \oplus x2 = x3 \rightarrow x2 \oplus x3 = x1 \wedge x1 \oplus x3 = x2$
A6. $x1 \oplus x2 = x3 \rightarrow \neg(\exists x4) x4 \oplus x3 = x1 \wedge x4 \neq x2$
A7. $|x1| = |x2| = |x3| = |x4|$
A8. $crypt(x1, x2) \rightarrow random(x2)$

A model[4] of this theory realizes the perfect cryptosystem.

The one-time pad is a model of this theory and even though it can offer a maximum security, 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.

[2]Not necessary the bitwise exclusive-OR operation.

[3]The predicate random(t) guarantees that all the numbers of the domain have the same probability to be taken to compose the message t and they are not correlated[14, 12].

[4]A model of a theory of the first order satisfies the axioms of the theory in the chosen domain[5].

2.2 ARC: Overview

Similarly to all the existing stream ciphers, ARC presents two main periods:

1. During the first period the *key* and the *keystream* are generated using a hash function[5] 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 $m_0 m_1 \cdots m_{n-1}$;
   ciphertext message c is a sequence of bits $c_0 c_1 \cdots c_{n-1}$;
   keystream is a pseudo-random sequence of bits $k0 k_1 \cdots k_{n-1}$;

   $$c_i = m_i \oplus k_i$$

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

3 Design of the Algorithm

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

3.1 Generation of the key

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

3.1.1 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 arbitray length and with k the key:

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

[5]The properties of the hash function will be faced in §4.2

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.

3.1.2 Stream cipher's key

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

$$key = F(k \| pad_1 \| F(k \| pad_2 \| k)) = HMAC_k(k)$$

where F is the hash function with properties required in §4.2.

3.2 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 $p : \{0,1\}^+ \to \{0,1\}^+$ be a function such that,

$$p(x) = LSB_n(x), \quad \frac{|x|}{2} \leq n \leq |x|$$

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

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

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 monodimensional matrix $\mathcal{M} = (m_1, m_2, \ldots, m_i)$ so that $q(x_i) = LSB_{m_i}(x_i)$, $\forall i \geq 1$.

The keystream is generated as follows:

$$y(1) = f\_hash(key);$$

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

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

$$\vdots$$

$$y(n) = f\_hash(p(y(n{-}1)) \| q(y(n{-}2)) \| \ldots \| q(y(n{-}(n{-}1))) \| key);$$

$$keystream = y(1) \parallel y(2) \parallel \ldots \parallel y(n);$$

Denoted with l the length of the hash function's output, the function p guarantees a good level of security for the stream cipher and at least $2^{\frac{l}{2}}$ different possible outputs of the hash function[6]. However, it's 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 $n = \frac{l}{2}$ and $m_1 + m_2 + \ldots + m_i = \frac{l}{2}$ (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 codomain(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 shouldn't be[7]), 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, as shown in the proof of Theorem 2 in §4.2.

4 Security Analysis

To demonstrate the effectiveness of the stream cipher, it is compared to the one-time pad theory. Moreover, it is proposed an analysis of the necessary properties to assume from the hash function for the stream cipher to be considered secure. An analysis of the key and the keystream is brought, too

4.1 One-time pad compare

The goal here is to show that ARC is a "model" of the one-time pad theory.

It is self evident that axioms A1...A7 are satisfied.

The analysis is done to show that A8 comes true by a relativistic interpretation. In fact, if the password is not known and the keystream can't be deduced or guessed without it so that the keystream is computationally indistinguishable from a truly random binary string, the keystream can be considered, in this context(and then interpretation), absolutely random as the one of the one-time pad cryptosystem, satisfying A8. Observe that ARC can never be information-theoretically

[6] In this case the outputs of the hash function are the pieces of the keystream.
[7] See §4.2.

secure as the one-time pad because the keystream is not truly random(even if computationally indistinguishable from a truly random one); it is only computationally secure.

Theorem 1 *The stream cipher generates a pseudo-random keystream computationally indistinguishable from a truly random string of bits only depending on the input password.*

Proof of Theorem 1: The keystream is composed by concatenated strings[8] of bits thanks the underlying keyed iterated hash function. For hypothesis the considered hash function produces pseudo-random strings of bits computationally indistinguishable from a uniform binary string[9] and the design of the stream cipher[10] guarantees at least $2^{l/2}$ different possible strings, where l denotes the length of the hash function's output. Therefore, for basic probability properties, the whole keystream is a pseudo-random string computationally indistinguishable from a truly random string of bits and only depending on the input password, because composed by concatenated pseudo-random substrings computationally indistinguishable from truly random strings and only depending on the input password by the key.

■

ARC is a model of the one-time pad theory in the above interpretation[11].

4.2 Hash Function Properties

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

Properties:

- **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).

[8] See §3.2.

[9] Check the hash function properties in §4.2.

[10] See §3.

[11] The approach is to consider the pseudo-random binary string as truly random if computationally indistinguishable from a truly random binary string, the seed is unknown and an exhaustive attack on the seed is not computable in acceptable time.

- **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 2 *Let f be the underlying iterated hash function of the stream cipher designed in §3. 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 hash function's properties 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 2:

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 isn't 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's preimage with probability 1 and minimum computational power, contradicting the stream cipher's above mentioned security.
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 isn't 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 stream cipher's above mentioned security.
3. **Collision resistance.** Let f be the underlying hash function of the *secure* stream cipher. Suppose, for absurd, that f isn't 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 stream cipher's keystream, if n has the same value of the length of the hash function's output and $\mathcal{M}$ is the banal matrix, it can happen that a very short cycle of bits build the keystream, as shown below:

$$y(1) = f(key);$$

$$y(2) = f(p(y(1)) \parallel key);$$

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

$$y(2) = y(1);$$

Suppose that:

$$\mathcal{M} = (0, 0, \ldots, 0) \Longrightarrow \forall x \; q(x) = \varepsilon$$

where ε denotes the empty string, then:

$$y(3) = f(p(y(2)) \parallel q(y(1)) \parallel key) = f(p(y(1)) \parallel key) = y(2) = y(1);$$

$$\vdots$$

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

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 doesn't 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 $\epsilon > 0$, recovering the keystream, contradicting the stream cipher's above mentioned security.

■

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

4.3 Stream cipher's key

The stream cipher's key 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 can't 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 dependecies of k_1 and k_2 can't 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.

4.4 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 $\lceil k/n \rceil$ known MAC-text pairs for the verification.
- **The Envelope Method.** This method combies 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[12] if a second preimage attack on the hash function is feasible[10], but as specified in §4.2, the hash function has to be collision resistant which implies the second preimage resistance.

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.

4.5 Keystream: A very lucky attacker

The keystream is computationally indistinguishable from a truly random string of bits, as claimed by Theorem 1. A collision attack, as the one discussed in the proof of Theorem 2, is not plausible when the underlying hash function is collision resistant(even weakly), but a similar scenario could still happen.

Denoted with f the hash function, l the length of the hash function's output[13] and k the stream cipher's key, suppose that:

$$p(x) = LSB_{x/2}(x),$$
$$\mathcal{M} = (0, 0, \dots, 0) \Longrightarrow \forall x\ q(x) = \varepsilon$$

The keystream y is the concatenation of different blocks of bits, as shown below:

$$y = \underbrace{y(1)_l \| y(1)_r}_{y(1)} \ \| \ \underbrace{y(2)_l \| y(2)_r}_{y(2)} \ \| \ \underbrace{y(3)_l \| y(3)_r}_{y(3)} \ \| \dots \| \ \underbrace{y(n)_l \| y(n)_r}_{y(n)} \quad (1)$$

[12]Even if theoretically it can be considered weak, an attack is still computationally infeasible.

[13]Observe that $l < 160$ bits are not considered, as required in §4.2.

where, $\forall i \geq 2 \quad y(i) = f(p(y(i-1))\|k) = f(y(i-1)_r\|k)$.

Now, since the blocks of the keystream contain a finite number of bits:

$$\exists i, j \wedge i \neq j : p(y(i)) = p(y(j)) \Longrightarrow y(i)_r = y(j)_r \tag{2}$$

When (2) occurs, substituting (2) in (1)(for some i and j) one can notice that the keystream is a cycle of the same bits from $y(i+1)$ to $y(j)$. If the "very lucky" attacker knows all blocks from $y(i)_r$ to $y(j)_r$ he can build the following keystream.

We defined the above attacker "very lucky" because of the low probability of (2) to happen, in fact:

Let be the event $E =$ "$p(y(i)) = p(y(j))$", then:

$$P[E] = \frac{2^{\frac{l}{2}}}{2^l} = \frac{1}{2^{\frac{l}{2}}}$$

Moreover, the adversary should know all or a significant number of blocks from $y(i)_r$ to $y(j)_r$ to realize a successful attack.

For this reason(and also to maximize the range of the hash function) in the design of the stream cipher it is strongly suggested to set $n = l/2$ and $m_1 + m_2 + \ldots + m_i = l/2$ (possibly not restricting only to m_1), drastically reducing the probability of such an attack.

5 Conclusions

An important aspect of this work is to consider the hash function as a black-box that satisfies the requirements analyzed in §4.2. 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.

ARC is unpatented. Anyone can use it at any time, in any way, royalty-free.

Kryptor[14] is a freeware tool implementing a sub-version of ARC-256 bits and can be found at http://www.rosiello.org under projects voice.

[14]http://www.rosiello.org/archivio/kryptor-0.1.2.tar.gz Kryptor uses MD5 as hash function that was recently broken, but it can't be considered vulnerable to those collision attacks since they should happen during the keystream generation and attackers can't control that phase.

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. Mendelson. *The language of first order logic.* Cambridge University Press, 1993.

[6] 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.