Encryption and Authentication in SECOQC

Jan Bouda[1][*], Oliver Maurhart[2][†], Thomas Themel[2][‡], Stephan Jank[3][§], Philipp Pluch[3][¶]
Rajagopal Nagarajan[1][‖]

[1] Department of Computer Science
The University of Warwick
Coventry CV4 7AL

[2]ARC Seibersdorf research GmbH
Donau-City straße 1, 1220 Wien, Austria

[3]Department of Mathematics
Klagenfurt University, Universitystreet 65-65
9020 Klagenfurt - Austria

April 30, 2007

1 Introduction

The goal of this document is to describe and analyze cryptographic algorithms and techniques used in SECOQC. The main priorities and evaluation points of view of are (in the descending order of importance):

- Security
- Efficient usage of key
- Time and space complexity

[*]bouda@fi.muni.cz
[†]oliver.maurhart@arcs.ac.at
[‡]thomas.themel@arcs.ac.at
[§]stephan.jank@uni-klu.ac.at
[¶]philipp.pluch@uni-klu.ac.at
[‖]R.Nagarajan@warwick.ac.uk

Directly following the motivation of the SECOQC project the security of the whole system is of crucial importance and we can use only unconditionally (information-theoretically) secure cryptographic primitives and techniques.

This determines the choice of the encryption algorithm to be the one-time pad. This algorithm is also optimal from the key consumption point of view[1] as proved by Shannon [8]. Finally, it is extremely efficient both from the time and the memory point of view, since it is a simple XOR of two bit strings - the message and the key.

The choice of the authentication algorithm is slightly less straightforward. The requirement of unconditional security (together with the key-efficiency point of view) restricts our choice to the Wegman-Carter [3] type authenticators. Let us suppose that we want to send n messages and that the probability of forging i–th message is at most p_i. It has been proven [3, 4] that in this case the length of the key is at least $\log_2 \frac{1}{p_1 p_2 \dots p_n}$. This limit can be asymptotically achieved by Wegman-Carter authenticators using a special technique proposed already in [3], we will discuss it in Subsection 4.2.

During the last twenty years there has been an extensive research in the area of Wegman-Carter authenticators aiming mainly the computational efficiency. To choose the most suitable authentication algorithm we followed results of Shoup, Nevelsteen and Preneel [9, 7]. We considered also two extremely fast algorithms develop after the tests of Shoup and Nevelsteen-Preneel, UMAC [1] and GMAC [6].

Following the results [9] and [7] we have chosen the evaluation hash to be the authentication algorithm for the SECOQC project. Main advantages are a reasonable key consumption (depending on the length of the message), an efficient computation and a possibility to authenticate messages of different lengths. Let us suppose that we divide the message into blocks of length l bits and that the maximal length of message is set to $n \times l$ bits. Then the probability of forgery for evaluation hash is appx. $n/2^{-l}$, while the asymptotical size of the key used per message is l bits.

Another possible candidate is the generalized division hash [5, 9], which should be implemented in a dedicated hardware device (most probably FCPGA PCI card) in case the performance of the software implementation of the evaluation hash turns out to be insufficient. Note that the question of efficiency of authentication algorithms from practical point of view might be complicated.

The evaluation hash is very efficient and easily exceeds 100 Mbits per second already when implemented in C without any manual assembler optimisation. This is by far exceeding SECOQC requirements, but the efficiency is necessary rather to minimize the load of the processor than to achieve an extreme speed of the authentication. The problem is that optimized versions of authentication algorithms are strongly dependent on an efficient usage of the internal processor cash and even processor registers. However, in real they will be running in a multiprocess environment, that can prevent an efficient usage of processor cash

[1]The length of the key must be equal to the length of the message, or, more formally, the entropy of the key must be at least as high as the entropy of the message.

and registers.

Therefore we leave open the possibility that a dedicated hardware will be used to implement the authentication algorithm. The generalized division hash seems to be more suitable for implementation in hardware.

Besides the two previous algorithms chosen according to tests of Shoup and Nevelsteen and Preneel, we also considered two algorithms proposed after publishing of the Shoup-Nevelsteen-Preneel tests – UMAC and GMAC. Both of them are extremely efficient, but their main purpose is computationally secure authentication. Therefore in the original proposal they considered usage of computationally secure cipher (e.g. AES) instead of one-time pad in the Wegman-Carter scheme for authentication of multiple messages, see Subsection 4.2. It can be easily replaced by one-time pad to achieve the unconditional security, however, the tag contributes to the security of the scheme only by half of its leng. It means that to achieve security 2^{-64} we have to use tag of size 128 bits and therefore we have to use 128 bits of key per message, what is inefficient. Therefore we decided not to consider these algorithms in the context of SECOQC.

2 Definitions of Security

In this subsection we briefly review basic approaches to the security of classical (and also quantum) cryptosystems.

The most strict type of security, called the unconditional (or information-theoretical) security, must hold without any assumptions. Especially it means that there is no way how the adversary can break the cryptosystem with probability higher than specified. In example, if we claim that the maximum probability of forging certain unconditionally secure authentication tag is 2^{-64}, then there is no way how to forge the tag with higher probability of success. In case the length of the tag is 64 (there are such systems) it means that there is no better way than to choose a random tag.

This type of security is the holy grail of cryptography, but there are only few cryptographic problems we can solve with unconditional security. Moreover, unconditionally secure solutions are usually inefficient namely from key usage and communication complexity point of view (surprisingly, there are usually computationally efficient). On the other hand, there is a number of problems that were proved to have no unconditionally secure solution.

In order to solve problems where no (suitable) unconditionally solution exists we introduce various assumptions. In this short introduction we discuss two such assumptions. First possible assumption is that the adversary is computationally limited, i.e. he has only limited time and resources to break the cryptosystem. Note that this assumption may be pretty reasonable - e.g. "the cryptosystem can be broken only using all computers of the Earth for 2^{64} years". Nevertheless, this is still an assumption since we can never be sure about actual computing power of our enemy. This type of security is called complexity-theoretic security. This assumption is usually formalized in the way that there is no BPP[2] algorithm able to break the cryptosystem.

The complexity-theoretic security is very reasonable, however, still rarely used in practice. Whenever we encounter practically used cryptosystem, we should expect that there are many more assumptions. The standard algorithms are only computationally secure - it means they are hoped to be complexity-theoretically secure. By the word "hoped" we mean that there is no known BPP algorithm breaking this cryptosystem, but it is not proven that there is none such. To ensure reasonability of the assumption we usually require that the cryptosystem is based[3] on some well–established mathematical problem, that remains intractable for a number of years.

The main motivation of SECOQC is to explore properties of quantum information to create communication with unconditionally secure encryption and authentication of messages.

3 Encryption

By encryption we understand a process of transforming the message m using knowledge of some secret information (key) k in the way that the new message c gives to the adversary no additional information about m (comparing to his a priori knowledge about m). Moreover, the original message can be easily reconstructed by anyone knowing c and k.

Encryption scheme, or cipher, is a five-tuple $(e, d, \mathbf{P}, \mathbf{K}, \mathbf{C})$, where $\mathbf{P}$ is the set of possible plaintexts (input messages), $\mathbf{K}$ is set of possible keys, $\mathbf{C}$ is set of possible ciphertexts (output messages) and $e : \mathbf{M} \times \mathbf{K} \to \mathbf{C}$ is the encryption function, and $d : \mathbf{C} \times \mathbf{K} \to \mathbf{M}$ is the decryption function satisfying the identity

$$\forall\ m \in \mathbf{M}, \forall k \in \mathbf{K}\ d(e(m, k), k) = m.$$

In the case of SECOQC we are interested only in unconditionally secure encryption. Therefore we use the one-time pad encryption system. Let us consider both the plaintex and the key to be binary strings $m[i]$ and $k[i]$ of the same length n. Encryption of the message $m[i]$ is described in Algorithm 1. Once the key is used, it must be discarded. It can be used neither for encryption of another message nor for authentication. Encryption of a message of length n consumes n bits of the key. Decryption is the same process as encryption except that the ciphertext c is the input and the message m is the output.

[2]Class of algorithms computable in polynomial time using randomized algorithm with one-side bounded error.

[3]The standard term "based" denotes another assumption. E.g. RSA is based on integer factorization problem, that is not known to be outside BPP. Moreover, solving integer factorization is sufficient to break RSA, but it is not known whether it is necessary or not.

Algorithm 1 One time pad

Input: message $m[i]$, key $k[i]$ (binary strings of length n)
Output: ciphertext $c[i]$ (binary string of length n)
Goal: Sender wants to send the message m to the receiver in a secure way.

```
1: for $i \leftarrow 0 \ldots n-1$ do
2:     $c[i] \leftarrow m[i]$ XOR $c[i]$
3: end for
```

4 Authentication

The authentication scheme is a five-tuple $(a, v, \mathbf{M}, \mathbf{K}, \mathbf{B})$, where $\mathbf{M}$ is the set of possible messages, $\mathbf{K}$ is the set of possible keys and $\mathbf{B}$ is the set of possible authenticated messages. There are two functions, the authenticating function $a : \mathbf{M} \times \mathbf{K} \rightarrow \mathbf{B}$ and the verifying function $v : \mathbf{B} \times \mathbf{K} \rightarrow \mathbf{M} \times \{accept, reject\}$. When Alice wants to send a message $m \in \mathbf{M}$ to Bob, she encodes it using the authenticating function to obtain $b \in \mathbf{B}$, $b = a(m, k)$, where $k \in \mathbf{K}$ is some secret key shared by Alice and Bob. When Bob receives b, he applies the function $v(b, k)$ to decide whether the message is authentic or not.

Regarding security of authentication schemes there are two basic security scenarios depending on what information the adversary has. In the weaker scenario the adversary knows only $\mathbf{M}$, $\mathbf{B}$, $\mathbf{K}$, a and v and tries to send some m' in hope that it will be accepted as authentic. This attack is usually denoted as **impersonation**.

More powerful is the attack when the adversary waits to receive an authentic message (a number of authentic messages) b and then replaces it with a different message while using possible advantage given by the knowledge of b. This attack is often denoted as **substitution**. In our definition we concentrate on the substitution when the adversary intercepted only single message. It is clearly at least as strong as impersonation (the adversary can always 'forget' the message he received), on the other hand more intercepted messages give no advantage[4] in the scenario of universal hashing since independent key is used for each message.

Considering the substitution attack the correctness and security of the scheme is defined as[5]

- If $b = a(m, k)$ for some $m \in \mathbf{M}$, then $Prob(v(b, k) = (m, accept)) = 1$.
- In case adversary replaces the message b by other message b', then $Prob(v(b', k) = (m', accept)) \leq \varepsilon$ for every $m' \in \mathbf{M}$ and a small nonnegative ε.

Note that $\varepsilon \geq 2^{-h}$, where h is the binary Shannon entropy representing uncertainty about the choice of the key $k \in \mathbf{K}$, since the adversary can always simply "guess" the key. In the case of SECOQC we require $\varepsilon \geq 2^{-64}$.

[4] We consider strongly universal$_2$ hashing in this paper.
[5] For more formal ways how to capture simultaneous correctness and security see e.g. [2].

The aim of the authentication is to prove the origin of the message and its integrity, i.e. to detect possible forgery and malicious modification of a message. It does not provide secrecy of a message.

Usually a special form of the previous definition is used. The authentication algorithm transforms the input message m into a pair $b = (m, t)$, where t is some **authentication tag** calculated using knowledge of m and k. Note that the original message is explicitly visible during the transmission and to verify authenticity of the message it suffices to verify whether it is accompanied by a correct tag.

4.1 Universal Hashing

The universal hashing is a concept introduced by Wegman and Carter [3] as a particular (class of) authentication scheme. Let $\mathbf{H} = \{h | h : \mathbf{M} \to \mathbf{T}\}$ be set of functions. In the context of authentication of messages, $\mathbf{M}$ is the set of messages and $\mathbf{T}$ is the set of authentication tags.

We say that a class of (hash) functions $\mathbf{H}$ is *strongly universal*$_2$ iff $\forall m_1, m_2 \in \mathbf{M}$, $m_1 \neq m_2$, and $\forall t_1, t_2 \in \mathbf{T}$, $t_1 \neq t_2$, it holds that the fraction of functions in $\mathbf{H}$ such that $h(m_1) = t_1$ and $h(m_2) = t_2$ is $|\mathbf{H}|/|\mathbf{T}|^2$.

This definition intuitively reformulates the security requirements for an authentication algorithm in terms of hash functions. Let us suppose that we have a strongly universal$_2$ class of hash functions and an authenticated message (m_1, t_1). From the definition it immediately follows that the fraction of functions fulfilling $h(m_2) = t_2$ in the set of functions $h \in \mathbf{H}, h(m_1) = t_1$ is $1/|\mathbf{T}|$ regardless of the values $m_1,\ m_2,\ t_1,\ t_2$. It means that regardless of what authenticated message (m_1, t_1) we receive it does not help us to find a second pair (m_2, t_2).

4.2 Multiple Messages and Universal Hashing

Strongly universal$_2$ classes of hash functions provide perfect security but are inefficient from the key usage point of view, i.e. the class of hash functions (and hence the key) is unnecessarily large comparing to $|\mathbf{M}|$ and $|\mathbf{T}|$. By an easy argument [3] we obtain that the theoretical lower limit of the key length is a function the probability of forgery (ε) and it turns out that this limit can be achieved, while the key length is asymptotically independent of the length of $|\mathbf{M}|$ and $|\mathbf{T}|$.

Let $\mathbf{H} = \{h | h : \mathbf{M} \to \mathbf{T}\}$ be a strongly universal$_2$ set of functions. Let us suppose that we have key formed by k_h specifying a hash function from $\mathbf{H}$ and by a sequence $k_1, \ldots, k_n$ such that $\forall i = 1 \ldots n\ k_i \in \mathbf{T}$. We calculate authentication tag for each m_i, $i = 1 \ldots n$, as $t_i = h_{k_h}(m_i) \oplus k_i$, where $\oplus$ denotes the bitwise XOR. To create the tag we use each time the same (secret!) hash function and encrypt its output with one-time pad using independent key for each message.

In the general case each message must contain a unique message ID to ensure the correspondence between messages and keys[6]. This system maintains security

[6]In our case we will use finite cyclically reused message ID of sufficient size guaranteeing

[3] of the underlying class of hash functions while the key size asymptotically approaches $\log_2 |\mathbf{T}|$ per message. This size of the key is also necessary since the adversary can always randomly choose the authentication tag.

4.3 ε–AXU Classes

The scheme described in the previous subsection achieves asymptotically optimal key usage using strongly universal$_2$ classes of functions, but it turns out that such solution is far from being optimal from the computational complexity point of view. Instead of strongly universal$_2$ it suffices to use almost XOR universal classes of hash functions, which are more efficient, and the security of the scheme described in the previous subsection is not decreased [5].

The class of functions $\mathbf{H} = \{h|h : \mathbf{M} \to \mathbf{T}\}$ is ε–AXU (almost XOR universal) iff for any $m_1, m_2 \in \mathbf{M}$ and any $t \in \mathbf{T}$ it holds that the fraction of functions with the property $h(m_1) \oplus h(m_2) = t$ is at most ε.

Therefore to implement an efficient unconditionally secure message authentication an AXU class of functions is exactly what we need.

5 Evaluation Hash

5.1 Advantages and Disadvantages

The evaluation hash is the main candidate for the hash function used in SEC-OQC. It has a number of properties making it very suitable for our purposes, namely:

- It is computationally efficient.
- The length of the message for authentication can vary in a wide range.
- It is possible to start the computation of an authentication tag before the complete message is available, i.e. only sufficiently long prefix is necessary.
- It has a good key consumption rate.

All these properties will described in more detail through this section.

5.2 Definition and the Length of the Tag

The evaluation hash views input (bit string) of length[7] $n \times l$ as a polynomial $M(y)$ of degree at most n over $GF(2^l)$. The hash key is a randomly selected

that the corresponding key is already replaced when the key ID is to be reused. Therefore the correspondence between the key and the message is preserved.

[7] If the message length is not a multiple of l we can introduce some padding e.g. by '0' till the necessary length. Note that in the structure of the message it should be explicitly readable what was the length of the padding (e.g. by specifying the length of the message), otherwise possible attacker can freely add or delete '0' at the end of the message in the range of l. In example he can replace $x_1x_2 \dots x_{l-3}$ by $x_1x_2 \dots x_{l-3}0$.

element $\alpha \in GF(2^l)$. The hash value (tag) is $h = M(\alpha)\alpha \in GF(2^l)$. This family of hash functions is $\varepsilon - AXU$ with $\varepsilon \approx n/2^l$.

In SECOQC we use this hash function in the Wegman-Carter scheme for authentication of multiple messages as described in Subsection 4.2. Formally the key will be split into two parts, $k^{(1)}$ and $k^{(2)}$. The key $k^{(1)}$ will be used in the hashing scheme as described above, i.e. $\alpha = k^{(1)}$. The second part of the key, $k^{(2)}$, will be used for the one-time pad encryption as described in Subsection 4.2. In the rest of this section we explain how to efficiently compute the value h.

The length of the tag should be selected as a function of the desired security ε and maximal length of a message, i.e. $n \times l$. We want to obtain security at least 2^{-64} and therefore $l \geq 64$. On the other hand, the maximal length of the message 1×64 is clearly insufficient and therefore length of the tag should be increased. Because of the 32-bit architecture of the computer platform used we decided to set the length of the tag to **96** bits allowing the maximal length of the message to be $\mathbf{2^{32}} \times \mathbf{2^{96}}$ bits with $\varepsilon = 2^{32}/2^{96} = 2^{-64}$.

5.3 General Mathematics

The length of the tag is one of the most important parameters both from security and pre-computation point of view. Any change of the tag length implies also change of the pre-computed tables, see below. In this description we show how to implement evaluation hash efficiently for the tag length $l = 64$ bits. The difference in evaluation when a different tag length is selected is minimal and will be explicitly described when necessary.

As described in Section A, 64 bit string (or l-bit string, in general) will be viewed as a member of $GF(2^{64})$ ($GF(2^l)$), i.e. Galois field having 2^{64} members. This field is implemented as polynomials of degree at most 63 over $\mathbb{Z}_2$, i.e. polynomials with coefficients 0 or 1. To learn more about efficient arithmetic in this field see Section A.

The tag length determines the Galois field and thus also the primitive polynomial that should be used. To keep computation efficient for a general tag of length l the primitive polynomial should be of the form $f(x) = x^l + f_0(x)$, where the degree of $f_0(x)$ is small. In the specific case of 64 bit tags it can be e.g. $f(x) = x^{64} + x^4 + x^3 + x + 1$. To find a reasonable primitive polynomial of the desired degree I recommend to use google and store the polynomials as a part of the algorithm.

5.4 Efficient Computation of the Hash Value $h(m)$

To compute the hash $h(m)$ we have to evaluate the polynomial $M(y)$ for the value $y = \alpha$ and the resulting number multiply by α. To do this efficiently we use Horner's rule. Let

$$M(y) = \sum_{i=0}^{n} m_i y^i. \tag{1}$$

To evaluate the hash using Horner's rule we will express $h(m)$ as

$$h(m) \equiv M(\alpha)\alpha \equiv ((((m_n\alpha + m_{n-1})\alpha + m_{n-2})\dots)\alpha + m_0)\alpha \pmod{f(x)}. \tag{2}$$

Parentheses here determine the order in which the expression will be evaluated. Note that all operations we have to use to compute $h(m)$ are multiplication and addition in $GF(2^l)$ since $\alpha, m_i \in GF(2^l)$. In the rest of this subsection we concentrate on the explanation how to implement these operations efficiently.

As follows from Section A, addition is a bitwise XOR of two bit strings of length l. When implementing multiplication we will use the fact that we do not use general multiplication, we use only multiplication by one fixed polynomial $\alpha \in GF(2^l)$.

We proceed with $l = 64$. Let us denote $\alpha \equiv a(x) \pmod{f(x)}$. First we have to create two pre-computed tables. The first table allows us to look up the value of the expression

$$v(x) \leftarrow v(x)x^8 \bmod f(x) \tag{3}$$

for all polynomials $v(x)$ of the degree less than 64. This table will have 256 entries and due to special form of the polynomial $f(x) = x^{64} + f_0(x)$ each entry will have only 16 bits. Entries of the table follow from the fact that

$$(u(x) + w(x)) \bmod f(x) = u(x) \bmod f(x) + w(x) \bmod f(x) \tag{4}$$

for any two polynomials $u(x), w(x) \in GF(2)[x]$. We can use this fact to express $v(x)a(x) = u(x) + w(x)$ such that $\deg u(x) < 64$ and $w(x) = v(x)a(x) - u(x)$. We obtain that $u(x) \bmod f(x) = u(x)$ and since $\deg v(x) < 8$ it holds that $\deg w(x) < 8$. Therefore all we have to store in the table are outcomes for different values of the polynomial $w(x)$. Multiplication by x^8 is easy to calculate since it is only a shift by 8 positions.

The second table allows to look up the value of the expression

$$v(x) \leftarrow v(x)a(x) \bmod f(x) \tag{5}$$

for all polynomials $v(x)$ of a degree less than 8. This table will have 256 entries and each entry will have 64 bits.

We will use these two tables to evaluate the product $a(x)b(x) \pmod{f(x)}$ for a general $b(x) \in GF(2^{64})$. Let us rewrite

$$b(x) = \sum_{i=0}^{7} b_i(x)x^{8i}, \tag{6}$$

where each $b_i(x)$ is a polynomial of a degree less that 8. To calculate the product $a(x)b(x) \pmod{f(x)}$ we set the polynomial $c(x) = 0$ and perform the calculation

$$\text{for } i \leftarrow 7 \text{ down to } 0 \text{ do } c(x) \leftarrow c(x)x^8 + b_i(x)a(x) \bmod f(x). \tag{7}$$

Computation when $l \neq 64$

The difference in case $l \neq 64$ is that we have to adjust suitably the degree of polynomials $b_i(x)$. The degree of the polynomial specifies tradeoff between the memory used for the lookup tables and the number of iterations of the cycle (7).

5.5 Computing the Hash Value with an Incomplete Message

In the context of SECOQC it might be useful to start the computation before a complete message is know, e.g. to improve the allocation of computational resources. This can be realized with the evaluation hash because of the structure of the Horner's rule. It is pretty obvious from Equation 2 that it is sufficient to have only a prefix of the message to start the computation, i.e. to start the computation of the Horner's rule. The computation can continue as soon as more message blocks (of size l bits) are received. Of course, it cannot be finished until the complete message is available.

A Arithmetics in $GF(2^l)$

The Galois field $GF(2^l)$ is set of all polynomials over $\mathbb{Z}_2$ modulo some primitive polynomial $f(x)$ of degree l. In other words, it is the set of all polynomials of a degree at most $(l-1)$ and the choice of a primitive polynomial $f(x)$ influences only operations (multiplication, to be more specific).

Let $a(x), b(x) \in GF(2^l)$ be two polynomials

$$\begin{aligned} a(x) &= \sum_{i=0}^{l-1} a_i x^i \\ b(x) &= \sum_{i=0}^{l-1} b_i x^i \end{aligned} \tag{8}$$

and let

$$f(x) = \sum_{i=0}^{l} f_i x^i, \tag{9}$$

where $a_i, b_i, f_i \in \mathbb{Z}_2$.

The addition modulo $f(x)$ works as the standard polynomial addition since it does not change the degree of the polynomial, i.e. $a(x) + b(x) \pmod{f(x)}$ is the polynomial

$$c(x) = \sum_{i=0}^{l-1} (a_i + b_i) x^i. \tag{10}$$

In order to perform the multiplication $a(x)b(x) \pmod{f(x)}$ we first calculate the polynomial

$$c(x) = \left(\sum_{i=0}^{l-1} a_i x^i\right)\left(\sum_{i=0}^{l-1} b_i x^i\right). \tag{11}$$

This is the outcome of the multiplication, but we have to determine the shortest representant (having the smallest degree) of this remainder class, i.e. we have to calculate the remainder of $c(x)$ after division by $f(x)$ – a polynomial $r(x)$ such that

$$c(x) = d(x)f(x) + r(x) \tag{12}$$

and $\deg r(x) < \deg f(x)$. This polynomial $r(x) \in GF(2^l)$ is the outcome of the multiplication. For an efficient implementation see e.g.

`http://algo.inria.fr/seminars/sem99-00/zimmermann.html`

or Donald Knuth's 'The Art of Computer Programming'. Note that in our case all non-trivial multiplications are avoided due to the precomputation.

References

[1] J. Black, S. Halevi, H. Krawczyk, T.Krovetz, and P. Rogaway. Umac: Fast and secure message authentication. In *Crypto '99*, pages 216–233, 1999. LNCS 1666, full version at www.cs.ucdavis.edu/ rogaway/umac/.

[2] R. Canetti. A unified framework for analyzing security of protocols. *Electronic Colloquium on Computational Complexity*, Report No. 16, 2001. continuously updated!

[3] J. L. Carter and M. N. Wegman. New hash functions and their use in authentication and set equality. *Journal of Computer and System Sciences*, 22:265–279, 1981.

[4] V. Fak. Repeated use of codes which detect deception. *IEEE transactions on information theory*, 25(2):233–234, ??

[5] Hugo Krawczyk. LFSR-based hashing and authentication. In *Crypto '94*, pages 129–139, 1994. LNCS 839.

[6] David A. McGrew and John Viega. Flexible and efficient message authentication in hardware and software. http://www.zork.org/, 2003.

[7] Wim Nevelsteen and Bart Preneel. Software performance of universal hash functions. In *Eurocrypt '99*, pages 24–41, 1999. LNCS 1592.

[8] C. E. Shannon. A mathematical theory of communication. *Bell System Tech. J.*, 27:379–423,623–656, 1948.

[9] Victor Shoup. On fast and provably secure message authentication based on universal hashing. In *Crypto '96, LNCS 1109*, 1996. www.shoup.net/papers (newer version).