Universally Unique Identifiers: How to *ensure* Uniqueness while protecting the Issuer's Privacy[*]

Martin Schaffer
System Security
Klagenfurt University (A)
m.schaffer@syssec.at

Peter Schartner
System Security
Klagenfurt University (A)
p.schartner@syssec.at

Stefan Rass
Transportation Informatics
Klagenfurt University (A)
stefan.rass@uni-klu.ac.at

Abstract *Universally Unique Identifiers (UUIDs) – standardized in ISO/IEC 9834-8:2005 – are widely used to uniquely identify entities in modern IT-systems. Apart from what promised in the standard, UUIDs are not guaranteed to be unique while preserving the issuer's privacy. In this paper we introduce a novel concept called collision-free number generation that can be used to locally generate UUIDs which are provably globally unique. Moreover, if the presented technique is instanced carefully, a poly-bounded adversary is not able to efficiently identify the issuer of a UUID. Our approach is efficient in terms of communication, time and space. As a by-product, it can be applied in other areas where collisions have to be avoided (e.g. key generation, pseudonym systems and interactive proofs).*

Keywords: universally unique identifiers, collision-free number generation, privacy, unique keys, pseudonyms.

1 Introduction

Uniquely identifying (physical or virtual) objects by means of a binary string (called identifier) is an intrinsic requirement for any modern IT-system. So, the main requirement for these identifiers is quite obvious: Within the life-time of the system, two different objects must not hold the same identifier. Otherwise, the system will not be able to distinguish between these objects, which may lead to system failures or security breaches.
As long as the objects stay in the range controlled by the issuing party, there is no problem at all. The employed unique identifiers can easily be generated. In the naive approach, we simply use a counter, which is incremented before issuing a new identifier. In a distributed environment, where several issuing parties generate identifiers for objects, which will be exchanged with other instances of the system, we need to enhance the mechanism proposed above. Now, each issuing party will hold a globally (or system-wide) unique identifier, which is concatenated to the counter in order to generate globally unique identifiers.
Unfortunately, uniqueness is not the only requirement. In many applications we need some sort of privacy protection in terms of anonymity and unlinkability. Anonymity in this scope means, that a malicious party cannot determine the issuer of a given identifier efficiently. Unlinkability means, that he cannot check, if two given identifiers have been generated by the same issuer.
ISO/IEC 9834-8:2005 [8], the standard for universally unique identifiers (UUIDs), is based on RFC 4122 [18], and addresses both, uniqueness and privacy protection. These UUIDs (or globally unique identifiers in Microsoft's implementation) can be found in a large number of applications. For instance, they are used as identifiers

- in programming languages (for objects),
- in the windows registry,
- in databases,
- in XML,
- in RPCs of COM and CORBA,

and are used for SOAP request messages. All versions of UUIDs, generated according to the RFC 4122, are 128 bit numbers. Depending on the used generation algorithm, different properties hold:
Algorithm 1 guarantees global uniqueness for UUIDs V1 if IEEE 802 MAC-addresses are used and none of them are cloned or manipulated. For the case that no network address is available, a randomly chosen address is used. Hence, global uniqueness is not guaranteed, unless only network addresses are used. In that case, however, privacy is not guaranteed, since the UUID does not hide the MAC-address.

[*]Originally published in the Proceedings of SAM'07.

Algorithm 2 derives UUIDs from unique names, but with the use of cryptographic hash algorithms (MD5 for UUIDs V3 [23] and SHA-1 for UUIDs V5 [2]). Hence, there is only a 'hope' for global uniqueness since collision can occur (cf. birthday paradox [19]). Privacy is guaranteed due to the one-way property of cryptographic hash functions.

Algorithm 3 creates UUIDs V4 from random numbers, i.e. collisions can again occur due to the birthday paradox. Privacy is guaranteed since no name or network address is involved.

Interestingly, none of the three algorithms meets the design goals: guaranteed global uniqueness, while *at the same time* preserving the privacy of its issuer. Nevertheless, uniqueness is promised in ITU-T Rec. X.667 (cf. page 7 in [7]):

"Three algorithms are specified for the generation of unique UUIDs, using different mechanisms to ensure uniqueness."

The question that we try to answer in this paper is, if we are able to find a solution to efficiently guarantee global uniqueness while preserving the privacy of the issuer of a UUID.
A simple solution could be to locally run a block-cipher in counter mode, i.e. encrypting a unique identifier (e.g. the MAC-address) concatenated to a counter which is incremented in every round. Then however, uniqueness is only guaranteed if every number generator uses the same master secret (or public key if asymmetric encryption is used). Obviously, such an approach leads to a privacy hole because once the master secret is known, the used identifier can be efficiently obtained. Hence, an additional design goal must be, that inverting one number generation by chance, must not lead to an efficient inversion of any other number generation.

Related Work: Apart from the algorithms discussed above, a rare number of approaches towards unique number generation exists: In [6], a solution has been proposed, where uniqueness is guaranteed by concatenating the time to the physical position of a device (obtained by use of the Global Positioning System GPS). This approach, however, is not very practical, since the GPS is not very precise and furthermore not always available (e.g. within buildings). Another solution is to locally generate globally unique private keys, proposed in [17]. The solution there, however, requires that each generator is provided with the same master secret. Once this master secret has been found, all keys can be linked to their originators. Improved approaches without the use of master secrets have been proposed in [25]. The solutions there output unique numbers which are either very large, or – if generated by the same instance – mutually linkable. None of these solutions guarantees uniqueness of (short) numbers while providing unlinkability without the use of master secrets at the same time.

Our Contribution: In this paper we introduce the concept of *collision-free number generation* which provides local generation of globally unique numbers without any communication (apart from the initialization process). Given such a number, a poly-bounded algorithm is not able to efficiently identify the corresponding generator. Our construction *provably* outputs unique numbers which constitutes an intrinsic advantage over the algorithms which are widely used to generate UUIDs. Moreover, UUIDs generated by our techniques are not efficiently linkable to their originator and hence the privacy of the issuer is preserved.

Road-Map: The remainder of this work is organized as follows: In section 2, we describe a general way, how a collision-free number generator (CFNG) can be designed. In section 3, an efficient implementation is given for the generation of UUIDs. It is then analyzed according to the requirements stated in section 2. In section 4, further applications of a CFNG are motivated and discussed. The paper closes in section 5 with an open problem.

2 Basic Construction

The problems stated in section 1 lead to the following requirements on the design of a CFNG:

R1 (Uniqueness): A locally generated number must be globally unique for a certain time-interval.

R2 (Efficiency): The generation process must be efficient regarding communication, time and space.

R3 (Privacy): Here we distinguish two cases:

1. *Hiding*: Given a generated number, a poly-bounded algorithm must not be able to efficiently identify the corresponding generator.
2. *Unlinkability*: Given a set of generated numbers, a poly-bounded algorithm must not be able to efficiently decide which of them have been generated by the same generator.

For efficiency reasons, an identifier-based approach is used. Every generator is (once) initialized with a globally unique identifier, which we denote by UI. The idea now is, to derive several unique numbers from UI, such that none of them is linkable to UI. Henceforth, l_x denotes the bit-length of x.

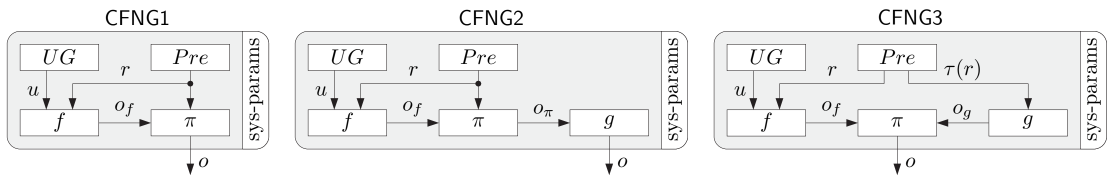

Figure 1: Three Variants for Collision-Free Number Generation

Uniqueness Generation: As a first step a routine is needed, which derives a unique number u from UI in every run of the generation process. We call such a routine *uniqueness generator* and denote it by UG. UG can be designed as follows:

$$u = UG(), \quad u = UI||cnt||pad$$

where pad is a suitable padding for later use of u and cnt is an l_{cnt}-bit counter initialized by $cnt \in_R \{0,1\}^{l_{cnt}}$ and incremented modulo $2^{l_{cnt}}$ in every round. So, the output of UG is unique for at least $2^{l_{cnt}}-1$ rounds. So far, the privacy is not preserved, since UI is accessible through u.

Uniqueness Randomization: A first step to provide privacy is to transform u such that the resulting block looks random. Hereby, we use an injective function f_r, where r is chosen from a set R. The idea is that u is randomized by r and hence we call f_r the *uniqueness randomization function*. We suggest to either use an injective one-way mixing-transformation for f_r according to Shannon [10] (e.g. symmetric encryption) or an injective probabilistic one-way function based on an intractable problem (e.g. the discrete logarithm problem [19]). In both cases, r is chosen at random. The output of f_r is obviously not guaranteed to be unique: Let u, u', $u \neq u'$ and r, r' random, where $r \neq r'$. Then $f_r(u) = f_{r'}(u')$ may hold since two different injective functions f_r and $f_{r'}$ map on the same output space. On the other hand, this problem cannot happen if $r = r'$, since f_r is injective. To generate a unique block o, sufficient information about the chosen function f_r has to be attached to its output. Hence, o can be defined as $o = f_r(u)||r$. The concatenation of two blocks by writing them in a row is an unnecessary restriction. The bits of the two blocks can be concatenated in any way. This leads to the following construction (cf. figure 1, CFNG1):

$$o = \pi(f_r(u), r), \quad u = UG(), \quad r = Pre()$$

where π is a (static) bit-permutation function (or bit-permuted expansion function) over the block $f_r(u)||r$ and the generation of r is done by a routine called *pre-processor*, denoted by Pre. The main-task of Pre is the correct selection of r. This can include a key generation process if f_r is an encryption function, for instance.

Theorem 2.1 Let u be a globally unique number, f_r be an injective function and $r \in R$. Furthermore, let π be a static bit-permutation function or bit-permuted expansion function. Then $o = \pi(f_r(u), r)$ is globally unique, for all $r \in R$.

Proof. Let $o' = \pi(f_{r'}(u'), r')$ and $u \neq u'$. The case where $r \neq r'$ obviously guarantees that the pairs $(f_r(u), r)$ and $(f_{r'}(u'), r')$ are distinct. Now consider the case where $r = r'$. Since $u \neq u'$ holds per assumption $f_r(u) \neq f_{r'}(u')$ holds due to the injectivity of f_r and the fact that $r = r'$. Thus, we have $(f_r(u), r) \neq (f_{r'}(u'), r')$ for all $r, r' \in R$. It remains to show that $o \neq o'$. This is obviously the case, because π is static and injective. □

If r is not sufficient to invert f_r, privacy protection is achieved through the one-way property of f_r. This can be the case if f_r is an asymmetric encryption function and r the public key. For the case that r can be used to invert f_r efficiently, we propose two extensions described in the following.

Privacy Protection: Given $o = \pi(f_r(u), r)$, computing $\pi^{-1}(o) = (a, b)$ is easy since π is public. Now assume that computing $f_b^{-1}(a) = u$ is feasible. To overcome this drawback we suggest using an injective one-way function g to hide $\pi(f_r(u), r)$. We call g the *privacy protection function*. The new output o is defined as follows (cf. figure 1, CFNG2):

$$o = g(\pi(f_r(u), r)), \quad u = UG(), \quad r = Pre()$$

The extended construction still outputs unique numbers, which is shown by the following corollary.

Corollary 2.1 Let u be a globally unique number, f_r be an injective function and $r \in R$. Furthermore, let π be a static bit-permutation function or bit-permuted expansion function and g an injective one-way function. Then $o = g(\pi(f_r(u), r))$ is globally unique, for all $r \in R$.

Proof. Let $o' = g(\pi(f_{r'}(u'), r'))$ with $u \neq u'$. By theorem 2.1 $\pi(f_r(u), r) \neq \pi(f_{r'}(u'), r')$. Since g is injective, $o \neq o'$ for all $r, r' \in R$. □

If f_r and g are chosen carefully, then o preserves the privacy, i.e. UI cannot be efficiently obtained from o. For particular applications it might be necessary, that only r and not $\pi(f_r(u), r)$ is hidden by g. To provide more flexibility we use $\tau(r)$ as input for g, where τ is a static bit-permutation function or bit-permuted expansion function. This leads to the following alternative (cf. figure 1, CFNG3):

$$o = \pi(f_r(u), g(\tau(r))), \quad u = UG(), \quad r = Pre()$$

The following corollary shows the uniqueness of o.

Corollary 2.2 Let u be a globally unique number, f_r be an injective function and $r \in R$. Furthermore, let π and τ be static bit-permutation functions or bit-permuted expansion functions and g an injective one-way function. Then $o = \pi(f_r(u), g(\tau(r)))$ is globally unique, for all $r \in R$.

Proof. Let $o' = \pi(f_{r'}(u'), g(\tau(r')))$ with $u \neq u'$. If $r = r'$ (resp. $r \neq r'$) then $g(\tau(r)) = g(\tau(r'))$ (resp. $g(\tau(r)) \neq g(\tau(r'))$) by the injectivity of $g \circ \tau$. Hence, theorem 2.1 can be applied through replacing r by $g(\tau(r))$ and so $o \neq o'$ for all $r, r' \in R$. □

The disadvantage of this approach is that in general the length of o is larger than in the previous construction. Moreover, even if $o \neq o'$ one can detect if $r = r'$ by the injectivity of $g \circ \tau$. In such a case the holder of r' (resp. r) can invert f_r (resp. $f_{r'}$). On the contrary, applying g only to r might be useful for scenarios where the issuer has to prove that o has a specific form.

Remark 2.1 Attention needs to be drawn on how g and f_r are instantiated. Since the inputs for both have a special structure, it has to be verified if the security provided by the functions is still preserved.

3 A Practical Approach

For the generation of UUIDs, we require the bit-length to be as short as possible without violating the defined requirements. CFNG1 is not very useful here (if r is sufficient to invert f_r) since UUIDs are public. CFNG2 outputs shorter blocks than CFNG3. Hence, we decided to use CFNG2. In the current implementation we use the existing MAC-address as the unique identifier UI (similar to algorithm 1 in the UUID-standard). In the following we propose an approach, where g is based on the elliptic curve discrete logarithm problem [21] to provide short output-lengths. For f_r ciphertext stealing [20] based on the SKIPJACK algorithm [3] is used.

3.1 Basic Tools

Elliptic Curve Cryptography (ECC): We assume that the reader is familiar with ECC (cf. [16]).

Definition 3.1 Let $E(\mathbb{Z}_p)$ be an elliptic curve group, where p is an odd prime. Let $P \in E(\mathbb{Z}_p)$ be a point of prime order q, where $q|\#E(\mathbb{Z}_p)$. The *Elliptic Curve Discrete Logarithm Problem (ECDLP)* is the following: Given a (random) point $Q \in \langle P \rangle$ and P, find $k \in \mathbb{Z}_q$ such that $Q = kP$.

By $SM(k, P)$ we henceforth denote the scalar multiplication kP in $E(\mathbb{Z}_p)$. It is believed that the ECDLP using $l_p \approx l_q \approx 160$ is secure against powerful attacks like Pollard's rho algorithm [16].

Point Compression [4]: A point on an elliptic curve consists of two coordinates and so requires $2l_p$ bits of space. It is clear that for every x-value there exist at most two possible y-values. Since they only differ in the algebraic sign, it suffices to store only one bit instead of the whole y-value. A point (x, y) can hence be stored as $x||b$, where $b = y$ MOD 2, and then only requires $l_p + 1$ bits of space.

Ciphertext Stealing: Let l_B be the block-length of a symmetric encryption function E. Let u be a plaintext, where $l_B < l_u \leq 2l_B$. If u is encrypted straight-forwardly by padding u up to $2l_B$ bits and then encrypting two blocks, the length of the corresponding ciphertext c is $l_c = 2l_B$. Using ciphertext stealing [20], c can be generated such that $l_c = l_u$. This works as follows: First u is cut into the blocks u_1 and u_2, where $l_{u_1} = l_B$ and $l_{u_2} = l_u - l_B$. Then u_1 is encrypted by use of E and a properly chosen key r resulting in a block $c_1||c_2$, where $l_{c_1} = l_u - l_B$ and $l_{c_2} = l_B - l_{c_1}$. Then the block $c_2||u_2$ is encrypted by use of E and the same key r resulting in the block c_3. This works, since $l_{c_2} + l_{u_2} = l_B$. The ciphertext of u is then $c_1||c_3$ and contains sufficient information to compute u, if r is available. The length of c is $l_c = l_{c_1} + l_{c_3} = l_u$. An example for a 64-bit block cipher can be found in figure 2.

3.2 Implementation

In the implementation shown in figure 2 we use the ECDLP and the point compression technique for the design of g. To obtain an output of 160 bit, we require $l_p = l_q = 159$, since 1 bit is needed to represent the y-value. We define g as follows:

$$g : \mathbb{Z}_q^* \rightarrow \mathbb{Z}_p \times \{0, 1\}, \quad [o_\pi] \mapsto [x, b]$$

where $(x, y) := SM(o_\pi, P)$ and $b := y$ MOD 2. Hence, $o_\pi = \pi(f_r(u), r)$ has to be an element of $\mathbb{Z}_q^*$. Since u is not random, we require $l_r = 80$ to

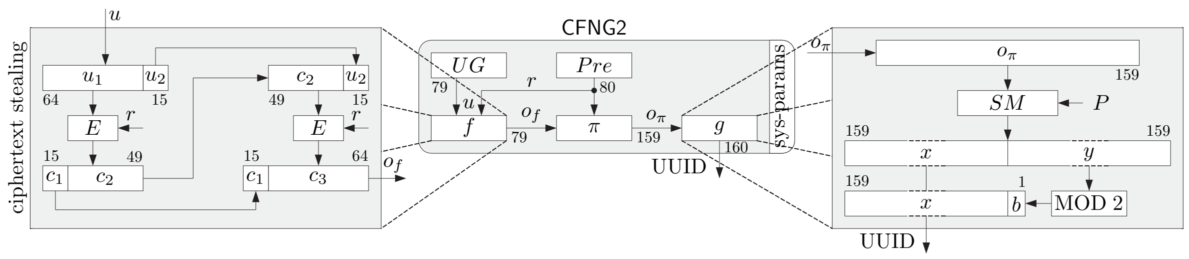

Figure 2: An Efficient Implementation

make brute-force attacks hard. To remain within a block of 159 bits, the output of f_r must be at most 79 bits. Since $l_{UI} = 48$, the counter can be at most 31 bits. Moreover, a verification is required to find out if $o_\pi \in \mathbb{Z}_q^*$ (this can be omitted if f_r outputs 78 bits, but then the counter has 30 bits). For efficiency reasons, we use a block-cipher for f_r. The SKIPJACK-algorithm (SA) [3] provides a key-length of 80 bits, which is useful, since we require $l_r = 80$. The SA operates on 64-bit blocks and thus we apply the technique of ciphertext stealing to encrypt u. We therefor set $l_B = 64$, $l_u = 79$ and hence get $l_{u_1} = 64$, $l_{u_2} = l_{c_1} = 15$ and $l_{c_2} = 49$. If $o_\pi \notin \mathbb{Z}_q^*$ then r is incremented modulo 2^{80} (to avoid an overflow) and the encryption process is applied to u again. After k runs of this procedure the probability for $o_\pi < q$ is $1/2^k$.

3.3 Analysis

Uniqueness: The implementation is an instance of CFNG2 and so uniqueness is guaranteed by corollary 2.1 if UG produces unique output. In the proposed implementation $l_{cnt} = 31$ and *cnt* is initialized by a random number. Thus *cnt* can be incremented modulo 2^{31} for $2^{31} - 1$ times without violating the uniqueness property.

Efficiency: No communication with third parties is necessary apart from the initialization process. The used ciphertext stealing requires two runs of the SA in every round. Scalar multiplication in $E(\mathbb{Z}_p)$ is efficient, since computations are performed in $\mathbb{Z}_p$, where p is only 159 bits long. Sometimes the generation process needs to be repeated, because it can happen that $o_\pi \notin \mathbb{Z}_q^*$. In this case, the process has to be repeated until $o_\pi \in \mathbb{Z}_q^*$ (by use of a new r). If we want to omit such a loop, we have to ensure that $l_{o_\pi} = l_q - 1$, which can be achieved by using a shorter counter. The length of a UUID is as short as possible, namely 160 bits (current security level). In general, UUIDs of lower length can be generated as well by some slight modifications, but then, either (strong) privacy cannot be guaranteed anymore, or a system-wide master secret has to be used (which we want to avoid).

Privacy (Hiding): The goal of an attack against the hiding property is to obtain UI, i.e. here the permanent 48-bit MAC-address. To obtain UI, the block u has to be found and then the counter removed. Therefore, o_π has to be accessible. If o_π looks sufficiently random, then the best known attack is Pollard's rho algorithm with a running time of $O(\sqrt{q/2})$, here $\approx 2^{80}$. Since we use a block cipher for f_r, a sequence of blocks of the form $f_r(u)$ appears random for a random r. Intuitively, the concatenation of the bits of a random number (here r) and a number that appears randomly (here o_f) results in a block which in-all appears randomly (here o_π). Since this is only a conjecture, we performed statistical tests over a generated sequence of 50 MB (generated by the described algorithm) using the NIST test suite [5]. The results are quite satisfying. A different attack could be that the attacker guesses that a specific MAC-address has been used. But then he has to mount a brute force attack to find o_f such that the given UUID equals to $g(\pi(f_r(u), r))$. Since $l_r = 80$, this attack is infeasible for a poly-bounded algorithm. Another approach could be to partially invert g to obtain the 80 bits of r. So far, no algorithm is known that achieves this attack goal. Intuitively, if an algorithm can obtain *any* selected bit, then it can find all bits and hence break the ECDLP.

Privacy (Unlinkability): Let o and o' be two UUIDs generated by the same generator. Then, the corresponding unique values u and u' at least differ in 1 bit. Since f_r is a mixing-transformation, changing 1 bit of the input results in $l_{o_f}/2$ bit-flips on average for the output o_f. Furthermore, r is chosen independently in every run and so half of the bits will be different on average. Hence, in total, o and o' differ in approximately one half of the bits. By the hiding property UI cannot be efficiently obtained from o or o'.

3.4 Extending the UUID-Standard

In the RFC 4122, UUIDs have been specified for a length of 128 bits. It is obvious, that we hardly achieve such a short block-length while fulfilling all the requirements stated in section 2. Based on our proposal the following two versions could be taken into account in the RFC 4122.

Identity-Based Short-Term Privacy: If the privacy of the generated UUIDs only needs to hold for a short period of time (for instance some days) then we suggest to modify our implementation such that $l_o = 128$. For instance, if $l_u = 64$ one can use the DES encryption algorithm for f, and then set $o = SM(\pi(DES(u, r), r), P)$, where $l_q = l_p = 128$ (here r would not contain parity-bits). Pollard's rho algorithm then has a running time of approximately 2^{64} steps, which is sufficient for a short time.

Identity-Based Long-Term Privacy: If privacy needs to hold for a long time (several years), we suggest to use an implementation like the one given in section 3. Then, however, additional lengths for UUIDs need to be specified in the RFC 4122.

4 Further Applications

In this section we exemplarily go through several further applications where the avoidance of collisions is intrinsic. Some of the applications are of current practical relevance, others can be interesting in the future. Henceforth, let $\mathbb{G}_q$ be a cyclic multiplicative group of prime order q, where the discrete logarithm problem (and related problems) are believed to be hard. Moreover, let $g \in \mathbb{G}_q \setminus \{1\}$.

4.1 Key Generation Algorithms

Key generation is based on (pseudo-)random number generators and hence collisions can occur. A collision can be dangerous if it is publicly detectable. In the context of private key cryptosystems this plays a role for remote controls contained in car-keys, for instance. In the following we discuss key-collisions in public key cryptosystems.

Discrete-Log based Cryptosystems: Several discrete-log (DL) based cryptosystems such as ElGamal encryption [13] use public keys of the form $e = g^d$, where $d \in_R \mathbb{Z}_q$. For two given public keys e and e' the equality $e = e'$ implies $d = d'$. Hence, the holder of d' can decrypt messages encrypted with e. Other DL-based schemes which are affected by a similar problem include [1, 11, 26]. This can be overcome, if DLs are generated by a proper CFNG.

RSA-based Cryptosystems: For cryptosystems that are based on the RSA-assumption [24], the following *must* not happen for two moduli $n = pq$, $n' = p'q'$, where p, q, p', q' are (safe) primes:

1. *Common Prime Attack:* If either p or q equals to p' or q' then the holder of n' can factor n.
2. *Common Modulus Attack [12]:* If $n = n'$ then w.l.o.g. $p = p'$ and $q = q'$. Hence, the holder of n' can factor n.

By the fundamental theorem of arithmetic, every natural number can be written as a product of prime-powers whose representation is unique. Hence, these attacks can be avoided if a substring (at fixed bit-positions) of each prime factor is chosen by an appropriate CFNG.

4.2 Probabilistic Encryption

A goal in the design of probabilistic public key schemes is providing indistinguishability of ciphertexts [15]: Given two ciphertexts $c_1 = E(m_1, e)$ and $c_2 = E(m_2, e)$ and the plaintext m_b, one must not be able to efficiently decide if $b = 1$ or $b = 2$. However, some schemes, such as ElGamal encryption, have the property that one can publicly detect, if two randomizers of different ciphertexts are equal, and then obtain partial or total information about the corresponding plaintexts: Let $(A, B) = (g^r, me^r)$ and $(A', B') = (g^{r'}, m'e^{r'})$ be two ciphertexts. From $A' = A$ it follows that $r' = r$ and hence $B'B^{-1} = m'm^{-1}$. If the message space is small, m and m' may then be found efficiently.

In the ElGamal signature scheme two colliding randomizers result in a much bigger problem: The secret key can be extracted. This problem also happens for several interactive proofs that are turned into signature schemes (cf. end of next section).

4.3 Interactive Proofs of Knowledge

Beside other properties, an interactive protocol is a proof of knowledge, if a so-called knowledge extractor (KE) can be given [9]. Such a KE informally works as follows: If a prover sends the same first message twice and correctly responds to two different challenges, then the secret information must be efficiently extractable. Obviously, in real protocol runs such a situation must not happen, otherwise a dishonest verifier can extract the secret. Consider Schnorr's proof of knowledge for instance: A prover wants to convince a verifier that he knows $x \in \mathbb{Z}_q$ for a given $y = g^x$. First he chooses $r \in_R \mathbb{Z}_q$ and sends $t = g^r$ to the verifier who returns a random challenge $c \in \mathbb{Z}_q$. The prover responds $s = (r - cx)$ MOD q and finally the verifier checks if $t = y^c g^s$ holds. The KE for this proof is the following: Given the transcripts (t, c, s) and (t', c', s') of two rounds, where $t = t'$, $c \neq c'$ and $s \neq s'$, the secret x can be obtained by computing $(s - s')(c' - c)^{-1}$ in $\mathbb{Z}_q$. Hence, if the prover accidentally selects $r = r'$ in any two runs of Schnorr's proof, the protocol-transcripts can be used to extract x. This problem can be overcome by generating r using a CFNG. The KE can still be given for the proof, but the extraction in practice is avoided. Notice, that the above stated problems also effect interactive proofs that are turned into a digital signature scheme using the techniques in [14].

4.4 Untraceable Devices

An application of a CFNG in this context can be the generation of temporary identifiers for the unique identification of RFID-tags. Since the generated identifiers are temporary, the RFID-tag is then not traceable anymore (if R3 is fulfilled).

4.5 Digital Pseudonyms

Digital pseudonyms are useful to protect one's privacy. The more 'fresh' (independent) pseudonyms are used (e.g. transaction pseudonyms) the lower is the linkability of different sessions of the same user. A core requirement is, that each pseudonym is globally unique [22]. This can be be achieved if a CFNG is involved (preferably type 2 or 3). Our identity-based approach might be utilized to provide *optional* anonymity revocation and linkability.

5 Future Prospects

An open problem is to instance our general construction by functions, where the privacy protection can be *provably* reduced to an intractable problem. So far, we have only been able to partly perform such reductions and hence had to perform statistical tests additionally.

Acknowledgements: The authors would like to thank Patrick Horster for his useful comments.

References

[1] FIPS-pub 186: Digital signature standard, 1994.

[2] FIPS-pub 180-1: Secure hash standard, 1995.

[3] NIST: SKIPJACK and KEA Alg. Spec., 1998.

[4] IEEE Std 1363-2000: IEEE Standard Specifications for Public-Key Cryptography, 2000.

[5] FIPS-pub 800-22: A statistical test suite for random and pseudorandom number generators for cryptographic applications, 2001.

[6] PriorArtDatbase, IPCOM#000007118D, 2002.

[7] ITU-T X.667: Generation and registration of Universally Unique Identifiers (UUIDs) and their use as ASN.1 object identifier components, 2004.

[8] ISO/IEC 9834-8: Generation and registration of Universally Unique Identifiers (UUIDs) and their use as ASN.1 Object Identifier components, 2005.

[9] M. Bellare, O. Goldreich. On defining proofs of knowledge. *LNCS 740:390–420*, 1993.

[10] C. Shannon. Communication theory of secrecy systems. *Bell Syst. Tech. Journ*, 28(4):656–715, 1949.

[11] R. Cramer, V. Shoup. A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. *LNCS 1462:13–25*, 1998.

[12] J. DeLaurentis. A further weakness in the common modulus protocols for the RSA cryptoalgorithm. *Cryptologia*, 8:253–259, 1982.

[13] T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *LNCS 196:10–18*, 1985.

[14] A. Fiat, A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. *LNCS 263:186–194*, 1987.

[15] S. Goldwasser, S. Micali. Probabilistic Encryption. *J. of Comp. and Syst. Scien. 28(2):270–299*, 1984.

[16] D. Hankerson, A. Menezes, S. Vanstone. *Guide to Elliptic Curve Cryptography.* Springer, 2004.

[17] P. Horster. Dublettenfreie Schlüsselgenerierung durch isoliert Instanzen. *In Chipkarten*, p. 104–119, Vieweg-Verlang, 1998.

[18] P. Leach, M. Mealling, R. Salz. A Universally Unique IDentifier (UUID) URN Namespace. *Request for Comments (RFC) 4122*, 2005.

[19] A. Menezes, P. van Oorschot, S. Vanstone. *Handbook of applied cryptography.* CRC Press, 1996.

[20] C. Meyer, S. Matyas. Cryptography: A new dimension in computer data security. *John Wiley & Sons*, pages 77–85, 1982.

[21] V. Miller. Use of elliptic curves in cryptography. *LNCS 218:417–426*, 1986.

[22] A. Pfitzmann, M. Köhntopp. Anonymity, Unobservability, and Pseudonymity – A Proposal for Terminology. *LNCS 2009:1–9*, 2001.

[23] R. Rivest. The MD5 Message-Digest Algorithm. *Request for Comments (RFC) 1321*, 1992.

[24] R. Rivest, A. Shamir, L. Adelman. A method for obtaining digital signatures and public-key cryptosystems. *Comm. ACM 21(2):120–126*, 1978.

[25] P. Schartner. *Security Tokens.* IT-Verlag, 2001.

[26] C.-P. Schnorr. Efficient signature generation by smart cards. *J. Cryptology*, 4(3):161–174, 1991.