A Light Number-Generation Scheme for Feasible and Secure Credit-Card-Payment Solutions

Francesco Buccafurri and Gianluca Lax

DIMET, University of Reggio Calabria
via Graziella, Località Feo di Vito, 89122 Reggio Calabria, Italy
bucca@unirc.it, lax@unirc.it

Abstract. Disposable-number credit card is a recent approach to contrasting the severe problem of credit card fraud, nowadays constantly growing, especially in credit-card-based e-commerce payments. Whenever the solutions cannot rely on a secure extra communication channel between cardholder and issuer, the only possibility is to generate new numbers on the basis of some common scheme, starting from secret shared initial information. However, in order to make the approach feasible, the computational load both on issuer and customer side should be minimized, also to reduce the cost of user-side devices, keeping yet an adequate security level. In this paper we present a disposable-number credit card scheme meeting the above goals, going a step ahead w.r.t. the state of the art.

1 Introduction

Credit card fraud is nowadays a serious problem whose dimension is constantly growing. Indeed, there are a number of techniques used by attackers to sniff (during both on-line and traditional transactions) the fixed credit card number used for authentication, and to use it for fraudulent payments. This is of course a direct consequence of the intrinsic weakness of the traditional credit card processing system, where the key used for authentication is long-term, semi-secret, transmitted over insecure channels, sometimes completely disclosed. Consider that credit cards are still widely used in e-commerce (as well as in other e-activities) since it often happens that alternative secure payment methods (like [15,10]) are not applicable or preferred.

A recent approach to contrasting the above problem is based on the concept of disposable-number credit cards. According to this scheme, issuer and customer agree on a number to use for the transaction, then they discard it, generate a new number for the next transaction and so on. This way, sniffing an authentication number during a transaction does not give the attacker any useful credential.

There are commercial [9,8] solutions based on the above scheme. Unfortunately, such solutions are either still insecure, when the cardholder gets the disposable number from the issuer Web site and authenticates itself by sensible data (like a standard credit card number) [16] or too expensive (and little friendly), when the generation of the new number is executed on board of a smart card, and the issuer provides the cardholder with an additional device capable of displaying the disposable number. In principle, the extra cost related to the additional device could be eliminated by exploiting simple software solutions to display disposable numbers, interfaced with standard smart-card drivers. Anyway, the cost of (secure) smart cards is not irrelevant, so that a real applicability of the above strategy could be strongly related with the possibility of decreasing significantly also the cost of the card. Another related problem is that the number generation scheme should be enough secure, because, differently from schemes based on an extra communication channel (typically the Web), where numbers can be generated randomly by the issuer, necessarily there will be a mathematical link between a given disposable number and the successor one. This mathematical link could be thus exploited by the attacker in order to predict new numbers on the basis of the past ones. As a consequence adequate efforts are necessary in order to design number generation schemes sufficiently robust, w.r.t. possible attacks. There are a number of research solutions [17,5] addressing the above problem. The first approach of this type is presented in [13] that generates a new authentication number by encrypting in a smart card a set of possible restrictions describing some elements of the transaction itself. Starting from the consideration that encryption is too expensive to be realistically used in this context, the authors of [17] propose the use of context free grammars (thus not relying on any cryptographic algorithm) to generate disposable credit card numbers. Context free grammars present the property that the generation and validation of strings belonging to a given language can be done in polynomial time, but it is unfeasible to find the grammar given only the strings generated by it, since any conjectured grammar may fail on a new input string. However, the authors do not give any suggestion about how context free grammars have to be generated. As a consequence, an unlucky generation of the grammar may allow an attacker to easily guess the grammar. Moreover, as stated in [18], there exists no theoretical result about how difficult is it to guess another string which belongs to the same language, thus showing the impossibility to guarantee the security of this technique. Also the authors of [5] propose a more efficient solution using cryptographic hash functions rather than encryption.

In this paper we do a step ahead. Indeed, we propose an authentication number generation scheme that is much computationally easier than previous approaches. It is based on a new fast non-cryptographic hash function as the core of the solution, and is highly secure too. The interesting thing is that the computation which the generation scheme is based on is implementable in hardware by very simple circuits whose cost is relevantly smaller than the traditional smart-card chip. Observe that the high computational lightness is a key issue also from the perspective of the issuer-side computational load. This is not irrelevant whether we think of a huge number of clients simultaneously using their credit card.

The rest of the paper is organized as follows. In the next section we present the conceptual basis of our proposal. The detailed definition of the approach is given in Section 3, where we deal with the elements composing the disposable-number

$$\begin{tikzcd}
s \arrow[d, "\boldsymbol{B}"'] & & \\
s_1 \arrow[r, "\boldsymbol{H}"] \arrow[d, "\boldsymbol{B}"'] & x_1 \arrow[r, "\boldsymbol{P}"] & AN_1 \\
s_2 \arrow[r, "\boldsymbol{H}"] \arrow[d, "\boldsymbol{B}"'] & x_2 \arrow[r, "\boldsymbol{P}"] & AN_2 \\
\cdots & \cdots & \cdots
\end{tikzcd}$$

Fig. 1. Number generation scheme

generation scheme. Section 4 deals with the security issues. An implementation of our proposal is provided in Section 5. Finally, in Section 6 we draw our conclusions. For space limitations the theorem proofs are not included in the paper.

2 Overview of the Proposal

The number generation scheme is based on the following elements:

1. an initial seed s, generated by the credit card issuer, consisting of a k-bit string;
2. a *basic* function $\mathcal{B}$ to obtain a k-bit string from another k-bit string;
3. a hash function $\mathcal{H}$;
4. a *projection* function $\mathcal{P}$;

The scheme to generate disposable numbers is shown in Figure 1. In particular, by computing $\mathcal{B}(s)$ we obtain a new seed s_1 that is recorded in place of s. Starting from s, it is possible to create a chain of values $s_1, ..., s_n$ such that $s_{i+1} = \mathcal{B}(s_i)$ for $i > 1$. Since the function $\mathcal{B}$ is reversible, such a chain can be traversed also backward, by considering that $s_i = \mathcal{B}^{-1}(s_{i+1})$. Thanks to such a feature, the chain values are not required to be pre-computed and stored, but they can be generated on the fly (the advantages resulting from this feature regard all the cases when the issuer has to check the validity of an already burnt number, like in case of refunds).

Once the new seed s_1 is generated, we compute $x_1 = \mathcal{H}(s_1)$. The one-way property of the hash function guarantees that the knowledge of x_1 does not give an attacker the possibility to guess s_1.

The last step, $\mathrm{AN}_1 = \mathcal{P}(x_1)$ is necessary to transform the value obtained by the hash function to a new credit card number. The following elements s_i, such that $i > 1$, and the corresponding ANs are obtained by iterating the above procedure.

An important issue regards the hash function to be used. We need to exploit its *one-way* property and this is the only property we required. On the contrary, the approach followed in [4] uses SHA-1 [7], a particular hash function provided with additional features (it must satisfy the *collision-resistant* property) and for this reason defined *cryptographic* hash function.

In our approach, this strong assumption is not necessary. We just require that given $x = \mathcal{H}(s)$, it is hard for the attacker to guess s from the knowledge of x (i.e., to invert the function by a brute-force attack). Observe that for a cryptographic hash function it is required the infeasibility of finding any y (even different from s) such that $x = \mathcal{H}(y)$.

In order to reach the goal of computationally *non-invertibility* (by generate and test approaches) of the hash function, a possible strategy is to design a hash function producing a large number of collisions and, thus, a sufficiently large research space. In our scheme, we can easily set parameters in order to obtain about 2^{450} collisions per value.

Note that our approach does not consist simply in the substitution of a weak hash function in place of a cryptographic one in a typical number generation scheme (like [5]). It is intuitive to understand that this would result in a very insecure approach just because of the weakness of the hash function itself (and because, in a scheme like [5], a new number is calculated as a hash of a seed that contains the number obtained at the previous step). We have designed thus a new weak hash function and, coherently, a new number generation scheme guaranteeing the security of the approach.

3 Number Generation Scheme

In this section we give the definitions of the elements composing the number generation scheme, that are the *basic function* $\mathcal{B}$, the *hash function* $\mathcal{H}$, and the *projection function* $\mathcal{P}$, we study some important properties of the functions $\mathcal{B}$ and $\mathcal{H}$, and, finally, we deal with the problem of the definition of the *initial seed* s. Observe that the choice of the basic function, the hash function and the projection function, cannot proceed orthogonally. Since the non-secret result is the composition of the three functions, we have to avoid that they are based on the same elementary operations, giving useful information to the attacker to proceed by crypto-analysis techniques. To prevent this, as we will explain in the following in this section, the basic function is based on string reverse and sum, the hash function is based on XOR and shift, and the projection function is based on scaling operations.

Before going into detail about the elements composing our scheme, we need some preliminary notations that will be used along the paper.

Notations. We denote by $x^k = (x_0, \ldots, x_{k-1})$ a k-bit string, where x_j, such that $0 \le j \le k-1$, represents the j-th bit. We denote by $\widetilde{x}^k = (x_{k-1}, \ldots, x_0)$ the k-bit-reverse string. Moreover, $x^k + 1$ denotes the k-bit string representing the number obtained by summing x^k thought as a binary number and 1, in 2^k-modulo arithmetic. For example, given $x^3 = 111$, $x^3 + 1$ represents the string 000, since $(111 + 001)(\mathbf{mod}\ 1000) = 000(\mathbf{mod}\ 1000)$. Moreover, we denote by

1^k (0^k, resp.) the k-bit string composed of all 1s (0s, resp.). Finally, let x^j be a j-bit string. We denote by $x^i x^j$ the $(i+j)$-bit string obtained by juxtaposing x^j to x^i.

3.1 The Basic Function

The basic function $\mathcal{B}$ allows us to generate the sequence of seeds used in the scheme. The function is defined as follows.

Definition 1. *Given a k-bit string s^k, then $\mathcal{B}(s^k) = \widetilde{s}^k + 1$.*

In words, $\mathcal{B}(s^k)$ is obtained by reversing the string s^k and, then, by summing 1 (modulo 2^k). This function allows us to have a new seed at each generation. Clearly, the period of this function should be as large as possible, hopefully 2^k (the upper bound), in order to have a negligible probability of re-generating a seed during a plausible life of a credit card. The next theorem ensures that the above goal is reached provided that k is odd.

Theorem 1. *Given a k-bit string r_0^k with $k \textbf{ mod } 2 \neq 0$, let R^k be the sequence $\langle r_0^k, \dots, r_{2^k-1}^k \rangle$ such that $r_i^k = \mathcal{B}(r_{i-1}^k)$ for $1 \leq i \leq 2^k - 1$.*

Then, it holds that $r_i^k \neq r_j^k$, for any i, j such that $0 \leq i < j \leq 2^k - 1$.

3.2 The Hash Function

As observed in Section 2, our proposal is based on the usage of a *weak* hash function. In particular, we refer to a hash function able to guarantee a *weak one-wayness*, obtained by the generation of a large number of collisions.

The first question is understanding if some already existing weak hash function can be used for our purpose. A good candidate could be CRC (Cyclic Redundancy Check) [3], a non-cryptographic hash function that is widely used in error-detection contexts, both for its effectiveness to detect many kinds of errors and for its efficiency, since a simple shift register circuit can be constructed to compute it in hardware [11]. Observe that CRC is much faster than cryptographic hash functions, even if it is computed in software[1]. However, we will see in Section 4 that CRC cannot be used in our scheme due to an intrinsic weakness which it suffers from. So we have to design a new weak hash function keeping the nice computational features of CRC but eliminating its weakness. Let us describe first how CRC works.

CRC is computed to produce a n-bit string, named *checksum*, starting from an arbitrary length string, called *frame*, such that also a slight change of the frame produces a different checksum. The checksum is computed as the rest of

[1] In order to test the efficiency of our proposal, we have performed some experiments comparing the efficiency of CRC (64 bits) computation with SHA-1. The experimental results show that CRC is one magnitude order faster than SHA-1. Indeed, computing 10^9 CRC hashes required about 300 seconds, whereas SHA-1 took about 3800 seconds.

the binary division with no carry bit (it is identical to XOR) of the frame, by a predefined *generator polynomial*, a $(n+1)$-bit string representing the coefficients of a polynomial with degree n. CRC is thus parametric w.r.t. the generator polynomial and for this reason there are many kinds of CRCs. For example, the most frequently used are CRC32 and CRC64 that generate a checksum of length 32 and 64 bits, respectively. Obviously, the higher the checksum length, the better the effectiveness of CRC in error detecting. Beside dependence on the generator-polynomial length, CRC is parametric w.r.t. the value of its coefficients. Consequently, the goodness of CRC strictly depends also on the latter parameter. Among the several existing CRCs, in the following we will refer to CRC64 whose generator polynomial is defined by the ECMA standard [2], since we argue that a 64-bit fully tested CRC offers satisfactory robustness features.

CRC satisfies the one-way requirement introduced in Section 2. Indeed, given a k-bit frame s^k (with $k > 64$) and its w-bit (with $w = 64$) checksum c^w computed by CRC, there are 2^{k-w} collisions, that is there exist 2^{k-w} k-bit strings s_i^k such that $\text{CRC}(s_i^k) = c^w$. We may vary k in order to increase the number of collisions generated by CRC to any value (for example 2^{450}) to the goal of making practically infeasible a brute-force attack attempting to find the original frame s^k. Moreover, its implementation easiness and efficiency make CRC very appealing to be used in this context.

Beside these nice features, CRC is not immune from malicious attacks exploiting its linearity w.r.t. XOR (this weakness has been widely documented in the literature and already exploited in some application contexts, like Wep [1,19]). In particular, it holds that $\text{CRC}(a \text{ XOR } b) = \text{CRC}(a) \text{ XOR } \text{CRC}(b)$, that is the checksum of the XOR of two numbers is equal to the XOR of the checksums of the two numbers. In our case, this property of CRC could be in principle exploited by an attacker to obtain the hash of the i-th seed of an user (i.e. $x_i = \text{CRC}(s_i^k)$) starting from the knowledge of (1) the hash of the j-th seed of the user and (2) the XOR between s_i^k and s_j^k (this attack is analyzed in Section 4).

We need thus to construct a hash function not suffering from the above problem, and preserving the other nice features of CRC. The idea is to apply a cyclic right shift to each seed before calculating the CRC value. But, clearly, the number of such shifts cannot be equal for each seed, otherwise the prediction described above can be identically applied. The solution we adopt is that the number of cyclic right shifts applied on a given seed s_i^k is equal to the number of 1s occurring in the seed itself. We denote by $\overrightarrow{s}_i^k$ the resulting k-bit string.

Now we are ready to define our hash function $\mathcal{H}$.

Definition 2. *Given a k-bit string s_i^k, then $\mathcal{H}(s_i^k) = CRC64(\overrightarrow{s}_i^k)$.*

3.3 The Projection Function

The numbers found on credit cards share a common numbering scheme. For a standard 16-digit credit card, the number consists of a single-digit major industry identifier (MII) (4 for Visa, 5 for MasterCard, and so on), a five-digit issuer identifier number (IIN), an account number (AN), and a single digit checksum

(C) computed by the Luhn algorithm [6]. Thus, given a major industry and a given issuer, for every users only digits from 7 to 15 can change. The projection function $\mathcal{P}$ transforms the 64-bit string x_i^{64} generated by $\mathcal{H}$ into a 9-digit number AN_i. Observe that a trivial implementation of such a function as $\mathcal{P}(x^{64}) = x^{64}$ **mod** 10^9 is not a good solution since (1) it cannot be easily realized via hardware because 10^9 is not a power of 2 and (2) the distribution of values so obtained is not uniform (values ranging from 0 to $(2^{64} - 1)$ **mod** 10^9 are more probable than the remaining ones).

To overcame this problem we have implemented the following solution. First, we introduce two notations. Given a k-bit string K, we denote by $[k]_{i,j}$ with $1 \leq i \leq j \leq k$ the sub-string of K obtained by keeping the $j - i + 1$ bits starting from the i-th left-most bit. For example, given $k = 1000$, $[k]_{1,2} = 10$. Given a decimal number number N, we denote by $[N]_j$ its j-th left-most digit. For example, given $N = 56789$, $[N]_2 = 6$.

Our solution requires a modulo-10 (decimal) counter C, initialized to 0. Initially, C is increased by 1 if the first bit (i.e. the most significant) of x^{64} is 1. The remaining 63 bits of x^{64} are partitioned in 9 buckets of 7 bits. The i-th bucket is used to set $[\mathrm{AN}]_i$, that is the i-th digit of AN. The i-th 7-bit bucket may assume a value v_i ranging from 0 to 127, and is partitioned again in 11 intervals. The first 10 intervals have size 12, whereas the last one 8. Let p_i be the index of the interval which v_i belongs to. If p_i is less than 10, then the value of the i-th digit of AN is set to p_i. Otherwise (i.e., p_i is either 10 or 11), $[\mathrm{AN}]_i$ is set to the value stored in C and C is increased by p_i (modulo 10). It is easy to show that this procedure results in a uniform distribution of ANs.

4 Security Issues

In this section we analyze the robustness of the proposed disposable number generation scheme with respect to a number of possible strategies followed by an attacker to guess the next credit card number. In our analysis we consider firstly the case of a brute force attack trying to find the current seed s_i starting from the sniffing of the last-used credit card numbers. Then we describe possible cryptanalytic attacks exploiting the knowledge of (more than one) consecutive credit card numbers. The analysis is done assuming that $k = 511$.

Consider a brute-force attack conducted knowing some credit card numbers used by the user (thus he knows some AN_i of our scheme). Clearly, he has to guess the source hash value (i.e. x_i of our scheme) starting from AN_i. Since the projection function maps in an uniform way all the 2^{64} x_i in the set of 10^9 AN_i, the probability of success is $(2^{64}/10^9)^{-1}$, that is about 2^{-34}. At this step the attacker has found 2^{34} potential x_i. Let suppose the attacker can detect the correct x_i among the potential 2^{34} values. Then, he must find the original seed s_i such that $\mathcal{H}(s_i) = x_i$. By repeating the above reasoning, we obtain that the attacker will find $2^{511}/2^{64} = 2^{447}$ potential solutions. Observe that, if the value $\overline{s}_i$ chosen by the attacker (among the 2^{447} found) differs from the actual s_i (i.e. the current seed of the fraud victim), then the probability that $\overline{AN}_{i+1} = AN_{i+1}$, where $\overline{AN}_{i+1}$ is the next credit card number obtained by $\overline{s}_i$ and AN_{i+1} is that one obtained by s_i, is $\frac{1}{10^9}$, that coincides with the probability of guessing a valid credit card number with no background knowledge. Thus, these results should discourage the attacker from trying to break the scheme by brute force attacks.

Now consider the case the attacker knows a sequence C of c consecutive credit card numbers spent by the victim. By a brute force the attacker should test $10^{c*9/2}$ seeds to find a seed $\overline{s}$ producing such a sequence C. Observe that, since our generation scheme produces a mapping between a set of 2^{511} bit strings and a set of 10^9 (i.e., about 2^{30}) numbers, till c is less than $511/30 - 1 \approx 16$, the probability of guessing also the next credit card number of the victim is again 10^{-9}. For higher c, this probability becomes 1 but the number of seeds to test is really too large (more than 10^{76}).

Finally, consider an attack based on the weakness of the CRC computation. In Section 3, we have noted that every two steps, the "noise" introduced by the reverse operation is *quasi*-cancelled. To understand how this could be exploited for an attack, we observe that when a seed s_i^k has both the left-most and the right-most bit 0 (i.e., every four steps), the attacker knows that s_i^k XOR $s_{i+2}^k = 10^{k-2}1$ (recall that, according to our preliminary notations, $10^{k-2}1$ denotes a k-bit string of the form $1 \cdots 1$, with $k-2$ 0s). Thus, the CRC of s_{i+2}^k is easily predictable by exploiting the above property. This behavior can be generalized also for other bit configurations. It is easy to see that if s_i^k is of the form $00 \cdots 01$, then we expect that the XOR with the seed generated two steps ahead is of the form $10^{k-3}11$. Again, if s_i^k is of the form $10 \cdots 00$, then we expect that the XOR with s_{i+2}^k is of the form $110^{k-3}1$. Finally, if s_i^k is of the form $10 \cdots 01$, then we expect that the XOR with s_{i+2}^k is of the form $110^{k-4}11$. This is a symptom of the alternating destructive effect of the reverse operation and, further, of the general invariance of the internal part of the seed, when the basic function is applied. Observe that this negative effect is maximum whenever the seed is palindromic, because the effect of the reverse is null also on a single step.

The next theorem gives us the probabilistic support that a quasi-random generation of the initial seed prevents this drawback for the entire credit card life.

Theorem 2. *Let t and k be two positive integers such that $t < 2^{\frac{k-4}{2}}$. Let s^k be a k-bit seed of the form $10c^j d^{k-4-2j} e^j 00$, where c^j and e^j are j-bit strings, d^{k-4-2j} is a $(k-4-2j)$-bit string containing at least one 0 and $j = \lceil log_2 t \rceil + 1$. It holds that the sequence $S^t = \langle s_0^k, \ldots, s_t^k \rangle$ such that $s_0^k = \mathcal{B}(s^k)$ and $s_r^k = \mathcal{B}(s_{r-1}^k)$ for $1 \leq r \leq t$ does not contain any seed of the form $10f^{k-4}01$, where f is a $(k-4)$-bit string.*

The theorem states that (i) fixing both the first and the last two bits of the initial seed (to 10 and 00, respectively), and (ii) ensuring that the seed contains an internal centered range whose bounds are distant $\lceil log_2 t \rceil + 1$ from the bottom (and the top) of the seed itself such that at least one 0 occurs in this interval, then it results that for at least t applications of the basic function (thus, at least for the next t credit card transactions), we do not generate bad seeds (i.e., seeds of the form $10 \cdots 01$). For example, in order to have the above property for the first $t = 50.000$ transactions, it suffices to set the initial seed to $10s_1^{17}s^{k-38}s_2^{17}00$, where s_1^{17}, s_1^{17} and s^{k-38} are randomly generated, with the only constraint that s^{k-38} contains at least one 0. It is easy to verify that the probability that a randomly generated string s^{k-38} does not satisfies the above requirement is $\frac{1}{2^{k-38}}$ (thus the blind random generation could be also accepted). For example in the case $k = 511$ this probability is $\frac{1}{2^{473}}$.

5 Implementation Issues

In this section we sketch the design of the hardware device implementing the number generation scheme so far described, in order to make evident that a strong positive point of our proposal is its feasibility and cheapness (especially w.r.t. other approaches based on smart card).

A concrete protocol implementing our scheme requires that the initial seed is generated by the credit card issuer, that is the provider of the device itself. In the following we assume the seed length is $k = 511$, that, as analyzed in the previous sections, guarantees a high security level. However there is no serious difficulty from the hardware point of view in further increasing this value in order to further hardening the system.

The device is equipped with three circuits implementing the basic function, the hash function and the projection function. The circuit implementing the basic function is composed of a 512-bit shift register R, storing the current seed and allowing the shift operation, and an adder used to implement the increment operation (for space reasons, we cannot describe the circuit).

Concerning the hash function implementation, it requires a simple shift register circuit and XOR gates (for details about CRC implementation and its faster table-driven implementation see the wide related literature [11,12,14]).

Finally, the projection function is implemented by means of a small combinatorial circuit having 7-bit input and 4-bit output (for example a ROM of $2^7 \cdot 4 = 512$ bits). This circuit works on each of the 9 buckets of 7 bits as described in Section 3.3, and returns the variable 9-digit number AN that, together with the fixed major industry identifier MII, the identifier number IIN, and the checksum C, produces the final disposal credit card number.

6 Conclusions

In this paper we have proposed a new number generation scheme used for CCT credit cards. The main contribution of our proposal is the simplicity of the computational machinery required to implement the scheme, resulting in a very simple and economic hardware implementation. It is well-known that the seeming low attention towards security aspects shown by issuers is actually the right compromise of a trade-off between costs to implement radical innovations and costs to refund customers victims of fraud. This explains why the aspects related to the practical feasibility of any proposed innovation to harden the credit card transaction processing is definitely important.

References

1. Borisov, N., Goldberg, I., Wagner, D.: Intercepting mobile communications: the insecurity of 802.11. In: MobiCom 2001: Proceedings of the 7th annual international conference on Mobile computing and networking, pp. 180–189. ACM Press, New York (2001)
2. ECMA. ECMA-182: Data Interchange on 12,7 mm 48-Track Magnetic Tape Cartridges — DLT1 Format (December 1992)
3. Hill, J.R.: A table driven approach to cyclic redundancy check calculations. SIGCOMM Comput. Commun. Rev. 9(2), 40–60 (1979)
4. Li, Y., Zhang, X.: A security-enhanced one-time payment scheme for credit card. In: Proceedings of the 14th International Workshop on Research Issues on Data Engineering: Web Services for E-Commerce and E-Government Applications (RIDE 2004), pp. 40–47 (2004)
5. Li, Y., Zhang, X.: Securing credit card transactions with one-time payment scheme. Electronic Commerce Research and Applications 4, 413–426 (2005)
6. Luhn, H.P.: Computer for verifying numbers. U.S. Patent 2, 950, 048 (1960)
7. NIST/NSA. Fips 180-2 secure hash standard (SHS). NIST/NSA (August 2002)
8. Dynamic passcode authentication, `http://www.visaeurope.com`
9. Private Payments, `http://www10.americanexpress.com`
10. Paypal, `http://www.paypal.com`
11. Peterson, W.W.: Error-correcting codes. MIT Press and J. Wiley & Sons (1961)
12. Ramabadran, T.V., Gaitonde, S.S.: A tutorial on crc computations. IEEE Micro. 8(4), 62–75 (1988)
13. Rubin, A., Wright, N.: Off-line generation of limited-use credit card numbers. In: Proceedings of the Fifth International Conference on Financial Cryptography, pp. 165–175 (2001)
14. Sarwate, D.V.: Computation of cyclic redundancy checks via table look-up. Commun. ACM 31, 1008–1013 (1988)
15. SET Secure Electronic Transaction LLC. SET Secure Electronic Transaction Specification, `http://www.setco.org`
16. Shamir, A.: Secureclick: A web payment system with disposable credit card numbers. In: Syverson, P.F. (ed.) FC 2001. LNCS, vol. 2339, pp. 232–242. Springer, Heidelberg (2002)
17. Singh, A., dos Santos, A.L.M.: Grammar based off line generation of disposable credit card numbers. In: SAC 2002: Proceedings of the 2002 ACM symposium on Applied computing, pp. 221–228. ACM Press, New York (2003)
18. Singh, A., dos Santos, A.L.M.: Context free grammar for the generation of one time authentication identity. In: FLAIRS Conference (2004)
19. Stubblefield, A., Ioannidis, J., Rubin, A.D.: A key recovery attack on the 802.11b wired equivalent privacy protocol (wep). ACM Trans. Inf. Syst. Secur. 7(2), 319–332 (2004)