"Pseudo-Random" Number Generation Within Cryptographic Algorithms: The DDS Case

Mihir Bellare[1], Shafi Goldwasser[2] and Daniele Micciancio[2]

[1] Dept. of Computer Science & Engineering, University of California at San Diego, 9500 Gilman Drive, La Jolla, California 92093, USA. E-Mail: mihir@cs.ucsd.edu. URL: http://www-cse.ucsd.edu/users/mihir.

[2] MIT Laboratory for Computer Science, 545 Technology Square, Cambridge, MA 02139, USA. E-Mail: {shafi,miccianc}@theory.lcs.mit.edu.

Abstract. The DSS signature algorithm requires the signer to generate a new random number with every signature. We show that if random numbers for DSS are generated using a linear congruential pseudorandom number generator (LCG) then the secret key can be quickly recovered after seeing a few signatures. This illustrates the high vulnerability of the DSS to weaknesses in the underlying random number generation process. It also confirms, that a sequence produced by LCG is not only predictable as has been known before, but should be used with extreme caution even within cryptographic applications that would appear to protect this sequence. The attack we present applies to truncated linear congruential generators as well, and can be extended to any pseudo random generator that can be described via modular linear equations.

1 Introduction

Randomness is a key ingredient for cryptography. Random bits are necessary not only for generating cryptographic keys, but are also often an integral part of steps of cryptographic algorithms. Examples are the DSS signature algorithm [16] which requires the choice of a new random number every time a new signature is generated, and CBC encryption, which requires the generation of a new random IV each time a new message is encrypted. (In fact, any secure, stateless encryption scheme must be probabilistic, requiring new randomness for each encryption [8].) In some cases, the random numbers chosen may have to be kept secret (as for DSS, where the leakage of one such random number compromises the secret key), whereas for other cases they can be made public (as in CBC encryption, where the IV may be sent in the clear).

In practice, the random bits will be generated by a pseudo random number generation process. For example, the DSS description [16] explicitly allows either using random or pseudo-random numbers. When this is done, the security of the scheme of course depends in a crucial way on the quality of the random bits produced by the generator. Thus, an evaluation of the overall security of a cryptographic algorithm should consider and take into account the choice of the pseudorandom generator.

B.S. Kaliski Jr. (Ed.): Advances in Cryptology - CRYPTO '97, LNCS 1294, pp. 277-291, 1997.

It has been well accepted that a good notion of pseudorandomness for cryptographic purposes is *unpredictability* [18,20,3,7]: given an initial sequence produced by a pseudo-random number generator on an unknown seed, it is hard to predict with better probability than guessing at random, the next bit in the sequence output by the generator. Such generators can be constructed based on number-theoretic assumptions, but are computationally costly. Alternatively, one could build a generator out of DES which would be unpredictable assuming DES behaves like a pseudorandom function, but in some contexts this may be deemed costly too, or we might not want to make such a strong assumption. Since using a weaker generator does not necessarily mean the resulting cryptographic algorithm is insecure, in practice one usually uses some weak but fast generator.

The intent of our paper is to illustrate the extreme care with which one should choose a pseudo random number generator to use within a particular cryptographic algorithm. Specifically, we consider a concrete algorithm, the Digital Signature Standard [16], and a concrete pseudo random number generator, the linear congruential generator (LCG) or truncated linear congruential pseudo random generator. We then show that if a LCG or truncated LCG is used to produce the pseudo random choices called for in DSS, then DSS becomes completely breakable.

We remark that the Standard [16] recommends the use of a pseudo-random generator based on SHA-1 or DES. The attack we describe does not say anything about the use of DSS with such generators, but it does illustrates the high vulnerability of the DSS to the underlying random number generation process.

We remark that LCGs are known to be predictable if part of the pseudo-random sequence is made public (see section 1.2 for details). However in DSS none of the pseudo-random numbers used is ever revealed, and thus predictability does not imply insecurity here.

Let us now look at all this more closely.

1.1 Pseudorandom numbers in DSS

Recall that the DSS has public parameters p, q, g where p, q are primes, of 512 bits and 160 bits respectively, and g is a generator of an order q subgroup of Z_p^*. The signer has a public key $y = g^x$ where $x \in Z_q$. To sign a message $m \in Z_q$, the signer picks at random a number $k \in \{1, \ldots, q-1\}$ and computes a signature (r, s), where $r = (g^k \bmod p) \bmod q$ and $s = (xr + m)k^{-1} \bmod q$.

Here the "nonce" k is chosen at random, anew for each message. In practice, a sequence of nonces will be produced by a generator $\mathcal{G}$ which, given some initial seed k_0, produces a sequence of values $k_1, k_2, \ldots$; k_i will be the nonce for the i-th signature.

The adversary (cryptanalyst) sees the public key y, and triples (m_i, r_i, s_i) where (r_i, s_i) is a signature of m_i. Notice that the secrecy of the nonces is crucial. If ever a single nonce k_i is revealed to the adversary, then the latter can recover the secret key x, because $x = (s_i k_i - m_i) r_i^{-1} \bmod q$. However, the nonces appear to be very well protected, making it hard to exploit any such weakness. The cryptanalyst only sees $r_i = (g^{k_i} \bmod p) \bmod q$ from which he cannot recover k_i short of computing discrete logarithms, and in fact not even then, due to the second mod operation. So even if $\mathcal{G}$ is a predictable generator, meaning, say, that given k_1, k_2 we can find k_3, there is no a priori reason to think DSS is vulnerable with this generator, because how can the cryptanalyst ever get to know k_1, k_2 anyway?

This might encourage a user to think that even a weak (predictable) generator is OK for DSS. This view would be wrong. We indicate that in fact DSS is vulnerable, because without a sufficiently good pseudorandom number generation process, the "masking" of the nonces provided by the algorithm is not sufficient to protect the nonces, even though recovering them seems a priori to require solving the discrete logarithm problem. In fact we prove a quite general lemma showing why this masking is essentially ineffective for pretty much *any* pseudorandom generator, and show specifically how to recover the keys when the generator is an LCG or truncated LCG. Thus one should not succumb to the temptation of using a weak generator for DSS.

1.2 Linear congruential generators

Recall that linear congruential generators are pseudo-random number generators based on a linear recurrence $X_{n+1} = aX_n + b \bmod M$ where a, b and M are parameters initially chosen at random and then fixed, and the seed is the initial value X_0. The advantage of linear congruential generators is that they are fast, and it has been shown [11] that they have good statistical properties for appropriate choices of the parameters a, b, M.

On the other hand, their unpredictability properties are known to be quite weak. Clearly they are predictable in their simplest form: if the parameters a, b and M are known, given X_0 all the other X_n can be easily computed. Plumstead (Boyar) [17] shows that even if the parameters a, b, M are unknown the sequence of numbers produced by a linear congruential generator is still predictable given some of the X_i. Truncated LCG were suggested by Knuth [12] as a possible way to make a linear congruential generator secure. However these generators have also been shown to be predictable [5,9,19] as have more general congruential generators [4,13].

However, as indicated above, this predictability does not directly mean a cryptographic algorithm using the generator is breakable, since it is possible none of the bits of the random numbers used by the algorithm are ever made public. DSS is (was) a case in point.

1.3 Cryptanalysis of DSS with LCG

DSS WITH LCG. We consider what happens when the nonces in DSS are generated using an LCG with known parameters a, b, M and hidden seed k_0. The predictability of the generator does not a priori appear to be a problem, due to the masking provided by the algorithm as indicated above. However, given just three valid signatures, we show how to recover the secret key.

UNIQUENESS LEMMA. We begin with a general lemma which indicates why the above intuition that the DSS protects the nonces may be false. The lemma (called the Uniqueness Lemma) says that as long as the nonces are pseudorandomly generated then, even if we *ignore* the relations $r_i = (g^{k_i} \bmod p) \bmod q$, the DSS signature equations $s_i k_i - r_i x = m_i$ uniquely determine the secret key with high probability. This means the cryptanalyst can effectively ignore the masking that is supposed to protect the nonces. This is true for *any* pseudorandom generation process, even a cryptographically strong one, using an unpredictable generator. This lemma tells us we can concentrate on the signature equations.

SOLVING THE EQUATIONS. We begin the cryptanalysis of DSS with LCG by combining the DSS "signature equations" with the LCG generation equations to get a system of equations. (In the process we ignore the $r_i = (g^{k_i} \bmod p) \bmod q$ relations, invoking the Uniqueness Lemma to say that solving the signature equations suffices to find the secret key.) However, this system is not trivial to solve because it is a system of simultaneous *modular* equations in *different* moduli. Techniques like Gaussian elimination fail. Instead we turn to lattice reduction. We show how to use Babai's closest vector approximation algorithm to solve such a system. The main difficulty here is dealing with the fact that this algorithm only returns (not very good) approximations to the closest vector. We then extend this to the case of the truncated LCG.

1.4 Other results, discussion, and implications

We extend our techniques to provide a general algorithm for solving a system of simultaneous linear modular equations in different moduli. (Another way of doing this, when the number of equations is constant, is to reduce the problem to integer programming in constant dimension and apply the algorithms of [14,10]. Our alternative solution seems simpler and more direct.)

In many cryptographic algorithms, the random numbers used are processed in a way that the public information gives little information about the original numbers. This is the case for the nonces in DSS. In such a setting, it may be reasonable to think that weak random number generators can suffice: even predictable generators could be fine because not enough information about the random numbers is revealed to make predictability even come into play. We are indicating this may not always be true: the quality of random bits matters even when the only thing an adversary sees is the result of a one-way functions on these bits.

A common pseudo-random number generator that comes standard with various operating systems is a linear congruential generator with modulus 2^{32}. It is plausible that there are DSA implementations available where the k values are formed by concatenating 5 consecutive outputs from such a generator. Our attack easily extends to this case.

2 Preliminaries

2.1 The Digital Signature Standard

The Digital Signature Standard (DSS, see [16]) is an ElGamal-like [6] digital signature algorithm based on the hardness of computing the discrete logarithm in some finite fields.

THE SCHEME. The scheme uses the following parameters: a prime number p, a prime number q which divides $p-1$ and an element $g \in Z_p^*$ of order q. (Chosen as $g = h^{(p-1)/q}$ where h is a generator of the cyclic group Z_p^*). These parameters may be common to all users of the signature scheme and we will consider them as fixed in the rest of the paper. The standard asks that $2^{159} < q < 2^{160}$ and $p > 2^{511}$. We let $G = \{ g^\alpha : \alpha \in Z_q \}$ denote the subgroup generated by g. Note it has prime order, and that the exponents are from a field, namely Z_q.

The secret key of a user is a random integer x in the range $\{0, \ldots, q-1\}$, and the corresponding public key is $y = g^x \bmod p$. DSA (the Digital Signature Algorithm that underlies the standard) can be used to sign any message $m \in Z_q$, as follows. The signer generates a random number $k \in \{1, \ldots, q-1\}$, which we call the *nonce*. It then computes the values $\lambda = g^k \bmod p$ and $r = \lambda \bmod q$. It sets $s = (xr + m) \cdot k^{-1} \bmod q$, where k^{-1} is the multiplicative inverse of k in the group Z_q^*. The signature of message m is the pair (r, s) and will be denoted by $\mathrm{DSA}(x, k, m)$. Note that a new, random nonce is chosen for each signature.

A purported signature (r, s) of message m can be verified, given the user's public key y, by computing the values $u_1 = m \cdot s^{-1} \bmod q$, $u_2 = r \cdot s^{-1} \bmod q$ and checking that $(g^{u_1} y^{u_2} \bmod p) \bmod q = r$. Notice that the values (r, s) output by $\mathrm{DSA}(x, k, m)$ satisfy the relation $sk - rx = m \pmod q$. We will make use of this relation in our attack on the DSS.

HASHING. The 160-bit "message" m above is not the actual text one wants to sign, but rather the hash of it, under a strong, collision resistant cryptographic hash function H. Specifically, if m is the actual text to be signed, the standard sets H = SHA-1, the Secure Hash Algorithm of [15]. The hashing serves two purposes. The first is to enable one to sign messages of length longer than 160 bits. Second, it "randomizes" the message to prevent any possible attacks based on the algebraic structure of the scheme. Accordingly, following [2], we treat the hash function as a random oracle.

We stress that we are considering *attacks*. In this context, treating H as a random oracle only strengthens our results. If the scheme is breakable when H is a random oracle, we should definitely consider it insecure, because a random oracle is the "best" possible hash function!

Our attack on the DSS algorithm does not involve the hash function H other than to assume it random. Therefore we will assume that the messages are already integers in the range $\{0, \ldots, q-1\}$ and that they are randomly distributed.

SECRECY OF THE NONCE. Recall that for every signature, the signer generates a new, random nonce k. An important feature (drawback!) of the DSS is that the security relies on the secrecy of the nonces. If any nonce k ever becomes revealed, at any time, even long after the signature (r, s) was generated, then given the nonce and the signature one can immediately recover the secret key x, via $x = (sk - m)r^{-1} \bmod q$. This is a key point in our attack.

2.2 Pseudo-Random Number Generators

Each time DSA is used to digitally sign a message m, a nonce k is needed. Ideally k should be a truly random number. In practice the nonces k are pseudo-random numbers produced by a pseudo-random number generator.

A pseudo-random number generator is a program $\mathcal{G}$ that on input a *seed* σ, generates a seemingly random sequence of numbers $\mathcal{G}(\sigma) = k_1, k_2, \ldots$.

The DSS algorithm can be used in conjunction with a pseudo-random number generator as follows. On input a secret key x, a seed σ to the generator, and a sequence of messages $m_1, \ldots, m_n$, run $\mathcal{G}(\sigma)$ to generate a sequence of pseudo-random numbers $k_1, \ldots, k_n$ and run DSA on input x, m_i, k_i for all $i = 1, \ldots, n$.

The pseudo-random number generators we consider in this paper are all variants of the linear congruential generator.

LINEAR CONGRUENTIAL GENERATORS. A linear congruential generator (LCG) is parameterized by a modulus M and two numbers $a, b \in Z_M$. The seed to $\mathcal{G}$ is just a number $\sigma = k_0 \in Z_M$. On input k_0, the generator produces a sequence of numbers, $\mathcal{G}(k_0) = k_1, k_2, \ldots$ defined by the linear recurrence $k_{i+1} = ak_i + b \bmod M$. The values k_i can be directly used by DSA as random nonces to sign the messages. (In which case they are treated modulo q. We assume that with high probability a k_i value will not be 0.)

TRUNCATED LINEAR CONGRUENTIAL GENERATORS. For security reasons it has been suggested that only some of the bits of the number produced by a linear congruential generator be used by applications, in our case the DSA algorithm. A truncated linear congruential generator does exactly this. Let's look at this more closely. A truncated linear congruential generator is parameterized by a modulus M, two numbers $a, b \in Z_M$ and two indices l, h such that $0 \leq l \leq h \leq \lg M$. The seed is a number $\sigma_0 \in Z_M$ and the generator produces numbers in the range $\{0, \ldots, 2^{h-l} - 1\}$. The generator computes a sequence σ_i according to the linear recurrence $\sigma_{i+1} = a\sigma_i + b \bmod M$. Then, each number σ_i is truncated by taking only bits $l, \ldots, h-1$ of the number, to get the number $k_i = ((\sigma_i - (\sigma_i \bmod 2^l)) \bmod 2^h)/2^l$ which is output by the generator.

3 The attack

We look at the security of the DSS when the nonces are generated using a LCG with parameters a, b, M. Later we will extend this to truncated LCGs.

3.1 Overview

Our attack on DSS exploits the relationship $sk - rx = m \bmod q$ holding for any digital signature $(r, s) = \mathrm{DSA}(x, k, m)$ produced by the DSA algorithm. The idea is this. Assume that we receive two messages m_1 and m_2 together with their digital signatures $(r_1, s_1) = \mathrm{DSA}(x, k_1, m_1)$ and $(r_2, s_2) = \mathrm{DSA}(x, k_2, m_2)$. We know that $s_1k_1 - r_1x = m_1 \bmod q$ and $s_2k_2 - r_2x = m_2 \bmod q$. The cryptanalyst knows $m_1, r_1, s_1, m_2, r_2, s_2$. He also knows the public parameters p, q, g of the DSS and the public key $y = g^x$ of the signer. What is hidden from him is the secret key x of the signer, and also the nonces k_1, k_2 which the signer used to produce the signatures.

At this point, the cryptanalyst is not expected to have any way of determining any of the unknowns short of computing discrete logarithms. However, now suppose we know that a linear congruential generator with parameters a, b, M has been used to produce the nonces. We assume the cryptanalyst knows the parameters a, b, M defining the LCG. (They were chosen at random, but then made public.) What is unknown to the cryptanalyst is the seed k_0 used by the signer to start the LCG. Now, we can combine the two signature equations above with the linear congruential equation $k_2 = ak_1 + b \bmod M$. These three equations together yield a system of three modular equations in three unknowns:

$$\begin{cases} s_1k_1 - r_1x = m_1 & \pmod{q} \\ s_2k_2 - r_2x = m_2 & \pmod{q} \\ -ak_1 + k_2 = b & \pmod{M} \end{cases} \tag{1}$$

Our approach is to try to solve these equations. Note it is a system of simultaneous modular linear equations in different moduli.

This approach at once raises two questions. One, of course, is how to solve such a system. But the other question may need to be addressed first. Namely, even if we solve it, how do we know the solutions we get are the desired ones? That is, there may be many different solutions, and finding a solution to the system (1) does not necessarily imply that we found the right one. (Meaning the one corresponding to the secret key x.)

This worry arises from a feature of this approach that we should highlight. We are not using all available information. We propose to ignore the fact that $r_i = (g^{k_i} \bmod p) \bmod q$. We will simply try to solve the equations, and see what we get. When we are ignoring what may seem a fundamental relation of the DSS signatures, it is not clear why solving the equations will bring us the right solutions: our system of equations might be under-determined.

We will answer this question in Section 3.2, showing that even disregarding the non-linear relationships $r_1 = (g^{k_1} \bmod p) \bmod q$ and $r_2 = (g^{k_2} \bmod p) \bmod q$, the solution to our equations is uniquely determined in most of the cases. Then we can turn to the problem of solving a system of modular linear equations.

If the moduli are the same, $M = q$, the equations can be easily solved by linear algebra. So, it is insecure to use q as the modulus in the LCG. However, if the modulus M is chosen (randomly and) independently from q, as we assume, one might still imagine that the equation $k_2 = ak_1 + b \bmod M$ does not help in finding the secret key because it is in a different modulus and cannot be easily combined with the other equations. In other words, we are faced with solving a system of simultaneous modular linear equations in different moduli. We address this via lattice reduction techniques in Section 3.3.

In later sections we extend the attack to truncated LCGs and also present a general method for solving systems of simultaneous linear modular equations in different modulii.

3.2 The Uniqueness Lemma

In this section we prove that when DSS is used with a pseudo-random number generator, a few signatures are usually enough for the linear equations $s_ik_i - r_ix = m_i$ to uniquely determine the secret key x, disregarding that it must be also that $r_i = g^{k_i} \bmod p \bmod q$. This answers the first question that we posed in section 3.1 and opens up the possibility of breaking DSS by solving a system of linear equations.

We stress this is true for any generator, not just LCG. The generator might be very strong (eg. cryptographically strong) or very weak, it does not matter. The number of signatures needed depends only on the length of the seed of the generator, growing linearly with this.

The statement we make is a probabilistic one: with high probability the system of equations obtained by using DSS with a linear congruential generator has a unique solution. The probability is taken over the choices of the messages to be signed only. (As discussed in Section 2, these are hashes of the real messages under some "strong" one-way hash function, and so considering them random is natural, especially from an attack point of view.) In other words, no matter how we had chosen x and σ, once they are fixed, if the messages m_i are randomly chosen the secret key x is uniquely determined with high probability.

Before stating the lemma we need some definitions. Fix a secret key $x \in Z_q$ of the DSS. Let $\mathcal{G}$ be some generator (not necessarily LCG) and let M be the total number of seeds that $\mathcal{G}$ can take. So we will think of a seed of $\mathcal{G}$ as being in Z_M. Now fix a seed $\sigma \in Z_M$ of the generator $\mathcal{G}$. Let $\mathcal{G}(\sigma) = k_1, k_2, \ldots, k_n$. Fix a message sequence $m_1, \ldots, m_n \in Z_q$ and let $(r_i, s_i) = \text{DSA}(x, k_i, m_i)$ be the signature of m_i using nonce k_i, for $i = 1, \ldots, n$. Let $x' \in Z_q$ and $\sigma' \in Z_M$, and let $\mathcal{G}(\sigma') = k'_1, \ldots, k'_n$. We say (x', σ') is a *false solution* with respect to $x, \sigma, m_1, \ldots, m_n$ if $x \neq x'$ but $s_ik'_i - r_ix' = m_i \bmod q$ for all $i = 1, \ldots, n$. That is, the secret key is not the right one, but the equations work out anyway.

Lemma 1. *Fix a secret key $x \in Z_q$ of the DSS, and a seed $\sigma \in Z_M$ of the generator $\mathcal{G}$. Now, choose n messages $m_1, \ldots, m_n$ uniformly at random from Z_q. The probability, over the choices of the messages only, that there exists some (x', σ') which is a false solution with respect to $x, \sigma, m_1, \ldots, m_n$ is less than Mq^{1-n}. Moreover, the expected number of such false solutions is also less than Mq^{1-n}.*

Proof. Let $k_1, \ldots, k_n = \mathcal{G}(\sigma)$ be the output of the generator on seed σ. Since σ is fixed, so are $k_1, \ldots, k_n$. We will assume these are all in Z_q^*.

Fix $x' \in Z_q$ and $\sigma' \in Z_M$ such that $x \neq x'$, and let $k'_1, \ldots, k'_n = \mathcal{G}(\sigma')$. For this fixed x', σ', and for a fixed i, we claim that the probability, over the choice of m_i, that $s_ik'_i - r_ix' = m_i$, is at most $1/q$. (Here $(r_i, s_i) = \text{DSA}(x, k_i, m_i)$, so that $r_i = g^{k_i}$ is a fixed quantity, while $s_i = (m_i + xr_i)k_i^{-1}$ is a random variable depending on the choice of m_i.) The reason this claim is not entirely obvious is that indeed s_i depends on m_i.

We first note that $s_ik'_i - r_ix' = m_i$ implies $k_i \neq k'_i$. To see this, note $m_i = s_ik_i - r_ix = m_ik'_i - r_ix'$. If $k_i = k'_i$ we would get $s_ik_i - r_ix = s_ik_i - r_ix'$ which yields $x = x'$ because $r_i \neq 0$. But we assumed $x \neq x'$.

Now, note that if $s_i k'_i - r_i x' = m_i$ then it must be that $(m_i + x r_i) k_i^{-1} k'_i - r_i x' = m_i$, or $m_i(1 - k_i^{-1} k'_i) = r_i x k_i^{-1} k'_i - r_i x'$. But $k_i \neq k'_i$ implies $1 - k_i^{-1} k'_i \neq 0$ whence

$$m_i = \frac{r_i(x k_i^{-1} k'_i - x')}{1 - k_i^{-1} k'_i} = \frac{r_i(x k'_i - x' k_i)}{k_i - k'_i} .$$

But the right hand side does not depend on the choice of m_i, because all the quantities there are fixed. (We use here that r_i does not depend on m_i, a property of DSS.) This means there is only one value m_i for which the above equation can be true. So if we pick m_i at random from Z_q, there is only a $1/q$ chance that the above equation can be true.

Now since the messages are chosen independently at random, the probability that $s_i k'_i - r_i x' = m_i$ for all $i = 1, \ldots, n$, is q^{-n}. Recall this is for fixed $x' \in Z_q$ $(x' \neq x)$ and $\sigma' \in Z_M$. The probability that there exists x', σ' which is a false solution is thus, by the union bound, at most $(q-1)M \cdot q^{-n} < Mq^{1-n}$. For the claim about the expected number of false solutions, use linearity of expectation instead of the union bound. ▌

Recall these results are true for any pseudo-random number generator $\mathcal{G}$. That is even if $\mathcal{G}$ is cryptographically strong, with high probability there will be only one secret key x and seed σ such that the equations $r_i x' + s_i k'_i = m_i$ are simultaneously satisfied. Clearly if $\mathcal{G}$ is cryptographically strong it will be hard to recover these x and σ from the signatures (r_i, s_i) and messages m_i only. But for the LCG it can be done.

3.3 Solving the equations

Lemma 1 shows that even if $M \neq q$, if M and q have the same size (i.e., $1/2 < M/q < 2$), the system of equations 1 will usually have only a few solutions. Therefore, if we can solve the system of equations we can also retrieve the secret key.

Solving via integer programming. We remark that systems of linear equations in different moduli can be rewritten as integer programming problems by introducing a new variable for each equation. Since we have a constant number of equations, they can thus be solved using polynomial time algorithms for integer programming in constant dimensions as given in [14,10]. However these algorithms are relatively complex and slow. Instead, we we want to solve more directly and simply. We now present a simple lattice based algorithm that solves our system using a nearest lattice vector approximation algorithm as a subroutine.

The nearest lattice vector problem. Let $B = \{b_1, \ldots, b_n\}$ be a finite set of vectors in $\mathbf{R}^n$. The lattice generated by B is the set of all integer combinations of the vectors in B and is denoted by $L(B)$. Given B and a vector $x \in R^n$ not in $L(B)$, the nearest lattice vector problem asks for a lattice vectors $w \in L(B)$ such that $\|w - x\| = \min_{v \in L(B)} \|v - x\|$. In [1], Babai gave a simple polynomial time algorithm to find an approximate solution to the nearest lattice vector problem: given the basis B and the target vector x, Babai's algorithm returns a lattice vector w such that $\|w - x\| \leq c \cdot \min_{v \in L(B)} \|v - x\|$, where $c = 2^{n/2}$ is an approximation factor depending only on the dimension of the lattice.

THE LATTICE. In order to solve the system of equations 1, we set up the following lattice. Let $x' = q/2$, $k_1' = k_2' = M/2$ and define also $\gamma_x = \min\{x', q - x'\}$, $\gamma_{k_1} = \min\{k_1', M - k_1'\}$, $\gamma_{k_2} = \min\{k_2', M - k_2'\}$. Consider the lattice L generated by the columns of the matrix

$$B = \begin{bmatrix} -r_1 & s_1 & 0 & q & 0 & 0 \\ -r_2 & 0 & s_2 & 0 & q & 0 \\ 0 & -a & 1 & 0 & 0 & M \\ \gamma_x^{-1} & 0 & 0 & 0 & 0 & 0 \\ 0 & \gamma_{k1}^{-1} & 0 & 0 & 0 & 0 \\ 0 & 0 & \gamma_{k2}^{-1} & 0 & 0 & 0 \end{bmatrix} .$$

Notice that multiplying the first three columns of the matrix by x, k_1, k_2 and subtracting the appropriate multiples of the remaining columns to perform modular reduction, we obtain the lattice vector

$$X = (m_1, m_2, m_3, x/\gamma_x, k_1/\gamma_{k1}, k_2/\gamma_{k2})^T$$

from which we can easily recover the secret key x.

Running Babai's nearest lattice vector algorithm on lattice $L(B)$ and target vector $Y = (m_1, m_2, m_3, x'/\gamma_x, k_1'/\gamma_{k1}, k_2'/\gamma_{k2})^T$ we obtain a lattice vector W such that $\|Y - W\| < 8\|Y - X\|$. Now, if $|x - x'| < \gamma_x/14$, $|k_1 - k_1'| < \gamma_{k1}/14$ and $|k_2 - k_2'| < \gamma_{k2}/14$, then $\|Y - W\| < 1$ and since the first three entries of W are integers they must coincide with the corresponding entries in Y and we have $W = (m_1, m_2, m_3, x''/\gamma_x, k_1''/\gamma_{k1}, k_2''/\gamma_{k2})^T$ for some x'', k_1'', k_2'' satisfying the equations 1. Moreover the following two inequalities are satisfied

$$x'' = (x'' - x') + x' \geq -\gamma_x + \gamma_x = 0$$
$$x'' = (x'' - x') + x' < \gamma_x + (q - \gamma_x) = q \,.$$

Inequalities $0 \leq k_1'', k_2'' < M$ can be proved analogously.

If the vector W does not have the desired form, that means that our initial guess (x', k_1', k_2') was not a good enough. If this is the case we simply repeat all the above steps with a different value for x', k_1', k_2'. One can check that if we let x' range in the set $\{q/2 \pm (1 - (1 - 1/8)^j)q/2 \mid j = 0, \ldots, 8 \lg q/2\}$, and k_1, k_2 in the set $\{M/2 \pm (1 - (1 - 1/8)^j)M/2 \mid j = 0, \ldots, 8 \lg M/2\}$, there will be some x', k_1', k_2' such that $|x - x'| < \gamma_x/14$, $|k_1 - k_1'| < \gamma_{k1}/14$ and $|k_2 - k_2'| < \gamma_{k2}/14$.

The number of possible x', k_1', k_2' to start with, to be sure of finding a solution to the system is polynomial in $\lg q$ and $\lg M$, so we can try all of them in polynomial time.

Once we have found a solution x', k_1', k_2' to the equations 1, we can check that we actually found the secret key x by computing $g^{x'} \bmod p$ and comparing it with the public key y. If $g^{x'} \bmod p \neq y$, then $x \neq x'$ and we did not found the solution that we wanted. In this case we can use the method just described to find a solution to the equations 1 in the range $0 \leq x < x'$ or $x' < x < q$. Since by Lemma 1 the total number of x such that system 1 has solution is less then

2 on the average, with high probability we will find the right x after one or two steps.

This completes the description of the attack to DSS when used with linear congruential generators.

4 Solving Simultaneous Modular Equations

The technique described in section 3.3 can be generalized to work on arbitrary systems of linear equations in different moduli. These kind of systems arises in the cryptanalysis of DSS when used with more sophisticated pseudo-random number generators, such as truncated linear congruential generators.

In this section we state the problem of solving a system of linear equations in different moduli in its full generality and give an algorithm to find a solution to such a systems. When the number of equations and variables is fixed, the running time of the algorithm is polynomial in the logarithms of all numbers involved in the description of the equations.

We consider the problem of finding "small" solutions to a system of modular linear equations in different moduli. More precisely, let $U_1, \ldots, U_n$ be positive integers and let V_U be the set of vectors $\{\boldsymbol{x} \in Z^n \mid \forall i.|x_i| < U_i\}$. Let also $A = \{a_{i,j}\}$ be an $m \times n$ integer matrix, and $\boldsymbol{b}$ and $\boldsymbol{M}$ be two vectors in Z^m. We want to find an integer vector $\boldsymbol{x} \in V_U$ such that $A \cdot \boldsymbol{x} = \boldsymbol{b} \pmod{\boldsymbol{M}}$, i.e., $|x_i| < U_i$ for all $i = 1, \ldots, n$ and the following modular equations are simultaneously satisfied

$$\begin{aligned} a_{1,1}x_1 + \ldots + a_{1,n}x_n &= b_1 \pmod{M_1} \\ &\vdots \\ a_{m,1}x_1 + \ldots + a_{m,n}x_n &= b_m \pmod{M_m}. \end{aligned}$$

We first assume that the above system has a solution x and that a good approximation to this solution is known and devise a method to find the exact solution.

Definition 2. Let $\boldsymbol{x}$ and $\boldsymbol{y}$ be two vectors in V_U. We say that vector $\boldsymbol{y}$ c-approximates $\boldsymbol{x}$ iff for all $i = 1, \ldots, n$ we have $|x_i - y_i| < (U_i - |y_i|)/(c\sqrt{n})$.

Lemma 3. *Let c be a constant greater than $2^{(m+n)/2}$. There exists a polynomial time algorithm that on input $U_1, \ldots, U_n, A, \boldsymbol{b}, \boldsymbol{M}$ as above and a c-approximation $\boldsymbol{y}$ to a solution $\boldsymbol{x} \in V_U$ to $A \cdot \boldsymbol{x} = \boldsymbol{b} \pmod{\boldsymbol{M}}$, finds a (possibly different) solution $\boldsymbol{w} \in V_U$ to $A \cdot \boldsymbol{x} = \boldsymbol{b} \pmod{\boldsymbol{M}}$.*

Proof. Let $\Gamma = \{\gamma_{i,j}\}$ be the $n \times n$ diagonal matrix defined by $\gamma_{i,i} = 1/(U_i - |y_i|)$ and let M be the $m \times m$ diagonal matrix whose diagonal entries are $M_1, \ldots, M_m$. Consider the lattice generated by the columns of the matrix $L = \begin{bmatrix} A & M \\ \Gamma & 0 \end{bmatrix}$ and define the vectors

$$\boldsymbol{X} = \begin{bmatrix} \boldsymbol{b} \\ \Gamma \cdot \boldsymbol{x} \end{bmatrix} \quad \boldsymbol{Y} = \begin{bmatrix} \boldsymbol{b} \\ \Gamma \cdot \boldsymbol{y} \end{bmatrix}$$

Notice that $\boldsymbol{X}$ is a lattice vector and

$$\|\boldsymbol{X}-\boldsymbol{Y}\| = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2\gamma_{i,i}^2} \leq \sqrt{\sum_{i=1}^{n}\frac{1}{c^2 n}} = \frac{1}{c}.$$

Running Babai's nearest lattice vector algorithm [1] on lattice L and target vector Y we obtain a lattice vector $\boldsymbol{W}$ such that $\|\boldsymbol{W}-\boldsymbol{Y}\| < c\|\boldsymbol{X}-\boldsymbol{Y}\| \leq 1$. Since the first m elements of $\boldsymbol{W}$ and $\boldsymbol{Y}$ are integers, they must be the same. So, the vector $\boldsymbol{W}$ is equal to $\begin{bmatrix} \boldsymbol{b} \\ \Gamma \cdot \boldsymbol{w} \end{bmatrix}$ for some integer vector $\boldsymbol{w}$ satisfying $A \cdot \boldsymbol{w} = \boldsymbol{b}$ (mod $\boldsymbol{M}$). It remains to be proved that $\boldsymbol{w} \in V_U$. Now for all $i = 1, \ldots, n$ we have

$$(w_i - y_i)^2\gamma_{i,i}^2 \leq \sum_i (w_i - y_i)^2\gamma_{i,i}^2 = \|\boldsymbol{W}-\boldsymbol{Y}\|^2 < 1,$$

so that $|w_i - y_i| < 1/\gamma_{i,i} = U_i - |y_i|$ and by triangular inequality

$$|w_i| \leq |y_i| + |w_i - y_i| < |y_i| + U_i - |y_i| = U_i.$$

This proves $\boldsymbol{w} \in V_U$.

We have shown how to solve a system of modular linear equations, given a good approximation to a solution. We now prove that for any fixed n and m there exists a set of vectors $D \subset V_U$ of size polynomial in the $\lg U_i$ such that for any $x \in V_U$ there exists a vector $y \in D$ such that y is a good approximation of x. This gives a polynomial time algorithm to solve modular linear systems in a fixed number of variables and equations.

Lemma 4. *Let $\delta > (1 + c\sqrt{n})/2$ and let D be the set $D_1 \times D_2 \times \cdots \times D_n$ where*

$$D_i = \{\pm(1 - (1 - 1/\delta)^j)U_i \mid j = 0, \ldots, \delta \lg U_i\}.$$

Then for any $\boldsymbol{x} \in V_U$ there exists a vector $\boldsymbol{y} \in D$ such that $\boldsymbol{y}$ is a c-approximation of $\boldsymbol{x}$.

Proof. Clearly it is sufficient to show that for all i and for all $x \in \{-U_i + 1, \ldots, U_i - 1\}$, there is some y in D_i such that $|x - y| < (U_i - |y|)/(c\sqrt{n})$. Since the set D_i is symmetric with respect to the origin, we can assume without loss of generality that $x \geq 0$. Now, notice that the sequence $y_j = (1 - (1 - 1/\delta)^j)U_i$ is increasing. Moreover $y_0 = 0$ and $y_{\delta \lg U_i} > U_i - 1$. Therefore there exists a $j \in \{0, \ldots, \delta \lg U_i - 1\}$ such that $y_j \leq x < y_{j+1}$. Now, let $x' = y_j + U(1 - 1/\delta)^j/(c\sqrt{n})$. If $x < x'$ then we have $c\sqrt{n}(x - y_j) < U_i(1 - 1/\delta)^j = U_i - y_j$ and $|x - y_j| \leq (U_i - |y_j|)/(c\sqrt{n})$. Otherwise $x \geq x'$ and we have

$$\begin{aligned} c\sqrt{n}(y_{j+1} - x) &< c\sqrt{n}y_{j+1} - c\sqrt{n}y_j - U_i(1 - 1/\delta)^j \\ &= U_i(1 - 1/\delta)^{j+1}(c\sqrt{n}/(\delta - 1) - 1) \\ &< U_i(1 - 1/\delta)^{j+1} = U_i - y_{j+1} \end{aligned}$$

and $|x - y_{j+1}| \leq (U_i - |y_{j+1}|)/(c\sqrt{n})$.

Theorem 5. *There is an algorithm which on input m modular equations in n variables and n positive integers $U_1, \ldots, U_n$, finds a solution $x_1, \ldots, x_n$ to the* *equations such that $|x_i| < U_i$ for all $i = 1, \ldots, n$ and for any fixed n and m the running time of the algorithm is polynomial in the sizes of the numbers.*

In the above theorem the interval in which the variables x_i ranges need not be centered around the origin, as if we want $L_i < x_i < U_i$ we can simply substitute $x_i - (U_i + L_i)/2$ for x_i and obtain an equivalent linear system to be solved in the interval $|x_i| < (U_i - L_i)/2$.

Corollary 6. *There is an algorithm which on input m modular equations in n variables and positive integers $L_1, U_1, \ldots, L_n, U_n$, finds a solution $x_1, \ldots, x_n$ to the equations such that $L_i < x_i < U_i$ for all $i = 1, \ldots, n$ and for any fixed n and m the running time of the algorithm is polynomial in the sizes of the numbers.*

5 Other Pseudo-Random Number Generators

In section 3.1 we presented an attack to DSS that involves the solution of a system of three modular equations in different moduli. The attack easily extends to any pseudo-random number generator expressible by modular linear equations. As an example we consider truncated linear congruential generator and generators where a long nonce is obtained by concatenating shorter random numbers.

5.1 Truncated Linear Congruential Generators

We recall that a truncated linear congruential generator computes a sequence of values σ_i starting from a seed σ_0 according a modular linear recurrence relation, and for each i outputs a number k_i obtained by taking the bits of σ_i between positions l and h. The computation of the σ_i can be easily be expressed by the equation

$$\sigma_i = a\sigma_{i-1} + b \pmod{M} \qquad (0 \leq \sigma_i < M) \tag{2}$$

where a, b and M are the parameters of the generator. Now let's look at how to express the truncation operation. If $l = 0$, then we can simply write

$$k_i = \sigma_i \pmod{2^h} \qquad (0 \leq k_i < 2^h). \tag{3}$$

If $l \neq 0$ we need two equations. First we extract the l-lowest order bits of σ_i via

$$d_i = \sigma_i \pmod{2^l} \qquad (0 \leq d_i < 2^l). \tag{4}$$

Then we use d_i to zero the l-lowest order bits of σ_i and extract the relevant bits of σ_i via

$$2^l k_i = \sigma_i - d_i \pmod{2^h} \qquad (0 \leq k_i < 2^{h-l}). \tag{5}$$

Notice that $\sigma_i - d_i$ is always an integer multiple of 2^l, so equation (5) has solution despite of the fact that 2^l has not an inverse modulo 2^h.

So, the entire process of computing $k_1, k_2, \ldots, k_n$ from a seed σ_0 can be expressed by modular linear equations (2),(4) and (5) for $i = 1, \ldots, n$.

Consider now the use of DSA with a truncated linear congruential generator $\mathcal{G}$ of parameters M, a, b, l, h. For concreteness, we assume that half of the bits are truncated, i.e., $h - l = (\lg M)/2$. Since we want to use the numbers output by the generator as nonces in the DSA algorithm, we also assume that $h - l = \lg q$. Consider the system of equations

$$\begin{cases} s_1k_1 - r_1x = m_1 & \pmod q \\ s_2k_2 - r_2x = m_2 & \pmod q \\ s_3k_3 - r_3x = m_3 & \pmod q \\ s_4k_4 - r_4x = m_4 & \pmod q \end{cases}$$

together with equations (2),(4) and (5) for $i = 1, \ldots, 4$.

By Corollary 6, we can find in polynomial time a solution to the above equations such that $0 \le x, k_i < q$, $0 \le \sigma_i < M$ and $0 \le d_i < 2^l$. By Lemma 1 we know that with probability $1 - M/q^3 > 1 - 2q^2/q^3 = 1 - 2/q$ the equations have no false solution. Therefore, with high probability, the solution x we found is the DSA secret key.

5.2 Linear Congruential Generators with Concatenation

If the numbers σ_i output by the pseudo-random generator are too short, the nonces k_i can be obtained by concatenating several σ_i together.

For example, a common pseudo-random number generator that comes standard with various operating systems is a linear congruential generator with modulus 2^{32}. The 160 bit number k required to sign with DSA can be obtained by concatenating 5 consecutive outputs from such a generator.

Our attack immediately applies to these schemes. Let σ_i be the sequence of random numbers defined by a linear congruential generator modulo $M = 2^{32}$.

The concatenation operation is easily expressed as a linear equation:

$$k_j = \sigma_{\alpha j} + M\sigma_{\alpha j+1} + \cdots + M^{\alpha-1}\sigma_{\alpha j+\alpha-1} \pmod{M^\alpha} \qquad (0 \le k_i < M^\alpha) \quad (6)$$

where $\alpha = \lceil \lg q / \lg M \rceil = 5$.

This time just two signature equations are enough to guarantee uniqueness of the solution with probability $1 - M/q \approx 1 - 2^{-128}$, and the secret key x can be easily found solving the system of modular equations

$$\begin{cases} s_1k_1 - r_1x = m_1 & \pmod q \\ s_2k_2 - r_2x = m_2 & \pmod q \end{cases}$$

together with equations (2) and (6) for $i = 1, \ldots, 2\alpha - 1$ and $j = 1, 2$.

The attack easily generalizes to generators involving any combination of truncation and concatenation operations.

Acknowledgments

The first author is supported in part by a 1996 Packard Foundation Fellowship in Science and Engineering, and NSF CAREER award CCR-9624439. The second and third authors are supported in part by DARPA contract DABT63-96-C-0018.

References

1. L. Babai. On Lovász' lattice reduction and the nearest lattice point problem. *Combinatorica*, 6(1):1–13, 1986.

2. M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. *Proceedings of the First Annual Conference on Computer and Communications Security*, ACM, 1993.
3. M. Blum and S. Micali. How to generate cryptographically strong sequences of pseudo-random bits. *SIAM J. Computing*, 13(4):850–863, November 1984.
4. Joan Boyar. Inferring sequences produced by pseudo-random number generators. *Journal of the ACM*, 36(1):129–141, January 1989.
5. A. M. Frieze, R. Kannan, and J. C. Lagarias. Linear congruential generators do not produce random sequences. In *Proc. 25th IEEE Symp. on Foundations of Comp. Science*, pages 480–484, Singer Island, 1984. IEEE.
6. Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In G. R. Blakley and D. C. Chaum, editors, *Proc. CRYPTO 84*, pages 10–18. Springer, 1985. Lecture Notes in Computer Science No. 196.
7. O. Goldreich, S. Goldwasser, and S. Micali. How to construct random functions. In *Proc. 25th IEEE Symp. on Foundations of Comp. Science*, pages 464–479, Singer Island, 1984. IEEE.
8. S. Goldwasser and S. Micali. Probabilistic Encryption. *Journal of Computer and System Sciences* 28:270–299, April 1984.
9. J. Hastad and A. Shamir. The cryptographic security of truncated linearly related variables. In *Proc. 17th ACM Symp. on Theory of Computing*, pages 356–362, Providence, 1985. ACM.
10. R. Kannan. Minkowski's convex body theorem and integer programming. *Mathematics of operations research*, 12(3):415–440, 1987.
11. Donald E. Knuth. *Seminumerical Algorithms*, volume 2 of *The Art of Computer Programming*. Addison-Wesley, 1969. Second edition, 1981.
12. Donald E. Knuth. Deciphering a linear congruential encryption. *IEEE Transactions on Information Theory*, IT-31(1):49–52, January 1985.
13. H. Krawczyk. How to predict congruential generators. In G. Brassard, editor, *Proc. CRYPTO 89*, pages 138–153. Springer, 1990. Lecture Notes in Computer Science No. 435.
14. H.W. Lenstra. Integer programming with a fixed number of variables. *Mathematics of operations research*, 8(4):538–548, 1983.
15. National Institute of Standards and Technology (NIST). *FIPS Publication 180: Secure Hash Standard (SHS)*, May 11, 1993.
16. National Institute of Standards and Technology (NIST). *FIPS Publication 186: Digital Signature Standard*, May 19, 1994.
17. J. Plumstead (Boyar). Inferring a sequence generated by a linear congruence. In *Proc. 23rd IEEE Symp. on Foundations of Comp. Science*, pages 153–159, Chicago, 1982. IEEE.
18. Adi Shamir. The generation of cryptographically strong pseudo-random sequences. In Allen Gersho, editor, *Advances in Cryptology: A Report on CRYPTO 81*, pages 1–1. U.C. Santa Barbara Dept. of Elec. and Computer Eng., 1982. Tech Report 82-04.
19. J. Stern. Secret linear congruential generators are not cryptographically secure. In *Proc. 28th IEEE Symp. on Foundations of Comp. Science*, pages 421–426, Los Angeles, 1987. IEEE.
20. A. C. Yao. Theory and application of trapdoor functions. In *Proc. 23rd IEEE Symp. on Foundations of Comp. Science*, pages 80–91, Chicago, 1982. IEEE.