The Prevalence of Kleptographic Attacks on Discrete-Log Based Cryptosystems

Adam Young* and Moti Yung**

Abstract. The notion of a Secretly Embedded Trapdoor with Universal Protection (SETUP) and its variations on attacking black-box cryptosystems has been recently introduced. The basic definitions, issues, and examples of various setup attacks (called Kleptographic attacks) have also been presented. The goal of this work is to describe a methodological way of attacking cryptosystems which exploits certain relations between cryptosystem instances which exist within cryptosystems. We call such relations "kleptograms". The identified kleptogram is used as the base for searching for a setup.

In particular, we employ as a discrete log based kleptogram a basic setup that was presented for the Diffie-Hellman key exchange. We show how it can be embedded in a large number of systems: the ElGamal encryption algorithm, the ElGamal signature algorithm, DSA, the Schnorr signature algorithm, and the Menezes-Vanstone PKCS. These embeddings can be extended directly to the MTI two-pass protocol, the Girault key agreement protocol, and many other cryptographic systems. These attacks demonstrate a systematic way to mount kleptographic attacks. They also show the vulnerability of systems based on the difficulty of computing discrete logs.

The setup attack on DSA exhibits a large bandwidth channel capable of leaking information which hardware black-box implementations (e.g., the Capstone chip) can use. We also show how to employ such channels for what we call "device marking".

Finally, note that it has been perceived that the DSA signature scheme was originally designed to be robust against its abuse as a public-key channel– to distinguish it from RSA signatures (where the signing function is actually a decryption function). In this paper we refute this "perceived advantage" and show how the DSA system (in hardware or software) can be easily modified to securely leak private keys and secure messages between two cooperating parties.

Key words: DSA signature, ElGamal encryption, ElGamal signature, Menezes-Vanstone PKCS, Schnorr signature algorithm, setup, Discrete-Log, Diffie-Hellman, subliminal channels, protocol abuse, kleptography, leakage-bandwidth, randomness, pseudorandomness, cryptographic system implementations.

* Dept. of Computer Science, Columbia University Email: ayoung@cs.columbia.edu.
** CertCo New York, NY, USA. Email: moti@certco.com, moti@cs.columbia.edu

1 Introduction

Recently, it has been shown that Black-Box cryptosystems can be designed so as to conform to public specifications and be polynomially indistinguishable from the known public specifications, and at the same time securely and subliminally leak secret key information to the implementor (either through keys at key generation or during run-time).

Young and Yung laid the foundation for these attacks, defined the basic notions, and demonstrated them [YY96, YY97]. These attacks imply that Black-Box systems (whose internals cannot be scrutinized) should not be automatically trusted (e.g., trust should be based on cryptosystems coming from a trustworthy source and not from the technology of tamper-resistant black-boxes, say).

Typically a cryptosystem produces a ciphertext for a given message or a signature for a given message. However, a cryptosystem with a setup produces a ciphertext/signature for a given message that also contains an internal ciphertext for a totally different message. We call such an output of a cryptosystem (with an inner ciphertext) a *kleptogram*. Kleptograms are undetectable in poly-time by the user, they are strong encryptions, and they coexist in the same ciphertext bits as normal public key encryptions.

In this paper a methodology for finding setup attacks is given. The methodology has two steps:

1. First we find a relation within a cryptographic function between its application and another inner encryption (this relation is called a kleptogram).
2. Then, given a cryptosystem and its workings, we identify how the kleptogram of the underlying function is embeddable in the system (and what leakage level is possible), which gives us a setup.

One of the setup attacks we present which is perhaps the main result of this work, is a (1,2)-leakage bandwidth setup for DSA. That is, we present a setup mechanism for DSA that is capable of leaking the user's private key through two (wlog) consecutive digital signatures. We then extend the attack to allow the user to send 160 subliminal bits of his choosing *in addition* to the private key. Furthermore, the user is free to re-key at any time and the attack will still work. The kleptographic attack therefore effectively leaks 80 key bits and 80 chosen bits per signature. This contrasts with the channel described by Simmons which leaks approximately 14 chosen bits per signature [Sim93]. Also, in the context of tamper-proof devices we show how the SETUP can be employed for "device marking", where the mark is added subliminally to the signature.

The above setup can be used to easily turn DSA into an effective public key (key exchange or message exchange) system. This spoofing, motivated by the potential of protocol abuse via kleptographic methods, shows that the claim that DSA is inherently different from RSA in this respect (the RSA signing function can obviously be used as a decryption function) is a myth! We refer the reader to the NIST response on DSA which alluded to this fact [SB92].

In this paper we show that these kinds of setup attacks are possible in other discrete log based cryptosystems such as the ElGamal encryption algorithm, ElGamal and Schnorr digital signatures, and various authenticated key exchange algorithms. Also, systems based on Elliptic Curves (Menezes-Vanstone) can be attacked. The attack methodology based on the discrete log kleptogram is therefore widely applicable to discrete log based cryptosystems.

2 Definitions and Background

A setup (Secretly Embedded Trapdoor with Universal Protection) is a mechanism that allows the secure leakage of private information within the output of a cryptosystem. The notion of a setup is due to Young and Yung [YY96]. The definitions of weak, regular, and strong setups and (m,n)-leakage are from [YY97].

Definition 1. Assume that C is a black-box cryptosystem with a publicly known specification. A (regular) SETUP mechanism is an algorithmic modification made to C to get C' such that:

1. The input of C' agrees with the public specifications of the input of C.
2. C' computes efficiently using the attacker's public encryption function E (and possibly other functions as well), contained within C'.
3. The attacker's private decryption function D is not contained within C' and is known only by the attacker.
4. The output of C' agrees with the public specifications of the output of C. At the same time, it contains published bits (of the user's secret key) which are easily derivable by the attacker (the output can be generated during key-generation or during system operation like message sending).
5. Furthermore, the output of C and C' are polynomially indistinguishable (as in [GM84]) to everyone except the attacker.
6. After the discovery of the specifics of the setup algorithm and after discovering its presence in the implementation (e.g. reverse-engineering of hardware tamper-proof device), users (except the attacker) cannot determine past (or future) keys.

Definition 2. A *weak setup* is a regular setup except that the output of C and C' are polynomially indistinguishable to everyone except the attacker and the owner of the device who is in control (knowledge) of his or her own private key (i.e., requirement 5 above is changed).

Definition 3. A *strong setup* is a regular setup, but in addition we assume that the users are able to hold and fully reverse-engineer the device after its past usage and before its future usage. They are able to analyze the actual implementation of C' and deploy the device. However, the users still cannot steal previously generated/future generated keys, and if the setup is not always applied to future keys, then setup-free keys and setup keys remain polynomially indistinguishable.

Another important notion is that of (m,n)-leakage bandwidth. A setup that has (m,n)-leakage bandwidth leaks m secret messages over the course of n messages that are output by the cryptographic device (or n of its executions).

Let us define a relation between a hidden encrypted value and another encryption/ signature.

Definition 4. A *kleptogram* is an encryption of a value (hidden value) that is displayed within the bits of an encryption/signature of a plaintext value (outer value).

Note that we say that a kleptogram is an encryption of a value, not a plaintext message. It is often the case in kleptography that the device is not free to choose this value. The device may calculate this hidden value, and then use it (for the 'randomness') in a subsequent computation, thus compromising that computation.

Related Work

The notion of a subliminal channel is due to Gus Simmons [Sim85, Sim93]. SETUPs exploiting such channels and a SETUP in the RSA public keys (as well as in other systems) were given in [YY96]. In [YY97] a SETUP was presented for Diffie-Hellman that does not make use of explicit subliminal channels, but rather exploits a number of executions of the system to "create a channel". This paper makes extensive use of this SETUP, which is a basic relation based on an underlying cryptographic problem. Due to its extensive applications as a relation between encrypted values, we call it the "discrete log kleptogram". The following section describes this relation in detail.

2.1 Discrete Log Kleptogram

Suppose that the only information that we are allowed to display is $g^c \ mod \ p$ for some $c < p-1$. The following is a way to leak a value c_2, over the single message $m_1 = g^{c_1} \ mod \ p$, such that the subsequent message $m_2 = g^{c_2} \ mod \ p$ is compromised. In this attack we assume that the device is free to choose the exponents used. Let the attacker's ElGamal private key be X, and let the corresponding public key be Y. Let W be a fixed odd integer, and let H be a cryptographically strong pseudorandom function with a hidden seed. WLOG, assume that H outputs values less than $\phi(p)$. The following algorithm is based on the operation of a Diffie-Hellman device that is used two times in succession. Let a and b be fixed constants.

1. For the first usage, $c_1 \in Z_{p-1}$ is chosen uniformly at random
2. The device outputs $m_1 = g^{c_1} \ mod \ p$.
3. c_1 is stored in non-volatile memory for the next time the device is used.
4. For the second usage, $t \in \{0, 1\}$ is chosen uniformly at random.

5. $z = g^{c_1 - WtY - ac_1 - b} \ mod \ p$.
6. $c_2 = H(z)$
7. The device outputs $m_2 = g^{c_2} \ mod \ p$.

The attacker need only passively tap the communications line, and obtain m_1 and m_2, in order to calculate c_2. We call z a *hidden field element.* The value for c_2 is found as follows.

1. $r = {m_1}^a g^b \ mod \ p$
2. $z_1 = m_1 / r^X \ mod \ p$
3. if $m_2 = g^{H(z_1)} \ mod \ p$ then output $H(z_1)$
4. $z_2 = z_1 / g^W$
5. if $m_2 = g^{H(z_2)} \ mod \ p$ then output $H(z_2)$

The value c_2 can be used by the attacker to determine the key from the second DH key exchange. Note that only the attacker can perform these computations since only the attacker knows X. In order for c_2 to be able to take on any value less $p-1$ we assume that $g_1 = g^{-Xb-W}$, $g_2 = g^{-Xb}$, and $g_3 = g^{1-aX}$ are generators mod p. This secure disclosure of an encrypted value inside the 'encryption' of another value will be exploited to attack a number of cryptosystems. As a side remark, note that technically nothing is encrypted in a DH key exchange. However, we may regard the resulting shared key as having been conceptually encrypted by both parties.

3 Setups in ElGamal Systems

Indeed, we are now ready to take the second step of our methodology, whereby we apply the above discrete log kleptogram mechanism to discrete log based cryptosystems. We start with applications to ElGamal, and then explain the more complicated setup attack on DSA in the next section.

3.1 Strong Setup in the ElGamal Encryption Scheme

In ElGamal [ElG85], the first part of the ciphertext of a message is $a = g^k \ mod \ p$. Note that a from the ciphertext (a, b) displays an exponentiation mod p. We can use this to implement a strong setup in ElGamal. In fact, the discrete log kleptogram *is* the strong setup for ElGamal encryption. It is straightforward to implement a (1,2)-leakage bandwidth scheme by leaking a hidden field element in the a of the first encryption (a, b), and using the hash of this element as the k for the second encryption. Note however that we are leaking messages m instead of private keys in this case. There is no known way to efficiently recover k from an ElGamal encryption, even if the user's private key x is known. So, it can be shown that this is a strong setup.

Theorem 1 *ElGamal encryption has a strong setup version.*

Typically, when public key cryptography is needed to encrypt bulk data, hybrid cryptosystems are used. Thus, in this mode of usage, the setup can leak keys. It can leak the randomly generated symmetric keys used to encrypt the data. We can implement a (1,1)-leakage scheme in an ElGamal based hybrid system as follows. We use the discrete log attack to setup up the $g^k \ mod \ p$ portion of the ciphertext. We then choose the one way hash of the hidden field element as all or part of the symmetric encryption key. It is therefore imperative to verify the source code of hybrid systems based on ElGamal.

3.2 Regular Setup in ElGamal Signature Scheme

In [YY96] an attack on the ElGamal signature scheme was proposed. This attack is novel in that it allows Alice to securely give her private key to Bob through signatures alone. However, the presence of the setup can be readily detected without knowledge of the attacker's ElGamal public key, and hence constitutes a weak setup.

The problem is that a user can always recover the choices of k of his own device using his own private key [Sim85]. Given the (wlog consecutive) ith and (i+1)th signatures, the user can compute $Y = (k_{i+1}^{-1})^{1/k_i} \ mod \ p$. After a few such coincidences, the user will conclude that Y is in fact the public key of the attacker. Note that the attack would still be very effective in hiding the key exchange from a warden (overseeing the communication) as in the original scenario of Gus Simmons. We can modify the attack to be a regular setup by using the fixed private parameters a, b, and W in conjunction with Y in the usual way, rather than simply setting $k_{i+1}^{-1} = Y^{k_i}$.

This setup attack can also be carried out using the discrete log attack. For concreteness, we will simply point out that the first ElGamal signature value $g^k \ mod \ p$ is a exponentiation mod p. Thus we leak a hidden field element as before, and this mechanism is a (1,2)-leakage setup. This kleptographic attack constitutes a regular setup. The fact that the value k_2 can't be compromised was shown in [YY97]. This assumes that the DH problem is hard. There is also the issue of detectability by the signer who knows k_1 and k_2. It can be shown that if H is a pseudorandom function whose seed is kept private by the implementor and hidden in the black-box device, the signer can't even detect the presence of the setup. The private values a and b are thus used as an extra precaution. This is not a strong setup, since a user knowing his own private key can recover the choices of k and detect the presence of the setup mechanism, given the seed, a, and b. The existence of a strong setup for the ElGamal digital signature scheme is left as an open problem.

Theorem 2 *The ElGamal digital signature algorithm has a regular setup version.*

4 SETUPing and Spoofing DSA

It has been assumed that the DSA [DSS91] system was designed as a signature system that is hard to "abuse". Namely, that it was designed so that it would not be used directly as public-key system, a key exchange system, or any system providing for confidential information exchange (see [SB92]). Therefore, it was quite interesting that a low bandwidth (14-bit) subliminal channel was found in it [Sim93]. Here we show a much larger leakage potential for black-box implementations of the DSA; we note that such implementations exist (i.e., the Capstone technology).

We will now briefly review DSA. q is a 160 bit prime which divides $p-1$. p is a prime that is at least 512 bits and at most 1024 bits in length. g is a qth root of 1 mod p. All three of these parameters are public. Alice's private key is x, where $x < q$. Alice's public key is y, where $y = g^x \ mod \ p$. Let H denote the Secure Hash Algorithm. To compute the signature (r, s) of a message m, Alice does the following.

1. chooses a value k at random such that $k < q$.
2. computes $r = (g^k \ mod \ p) \ mod \ q$.
3. computes $s = k^{-1}(H(m) + xr) \ mod \ q$.
4. outputs the signature (r, s).

To verify that the signature is valid, Bob checks to make sure that r is equal to $(g^{s^{-1}H(m)} y^{s^{-1}r} \ mod \ p) \ mod \ q$.

It is clear from the discrete log kleptogram that we need only find a modular exponentiation ($mod \ p$) that is displayed in DSA to find a setup. Note that a modular exponentiation $mod \ q$ won't suffice since q is too small. We could setup DSA keys, since $y = g^x \ mod \ p$, but this indicates that the user must generate new keys (or a new signature) to be an effective setup attack for the attacker. The existence of the modular exponentiation that constitutes the (1,2)-leakage setup for signatures is indeed a bit more subtle. But, it turns out that the quantity $g^{s^{-1}H(m)} y^{s^{-1}r} \ mod \ p$ is in fact simply $g^k \ mod \ p$, and can thus be used as a kleptogram. It follows that over the course of two (wlog) consecutive signatures, a DSA device can securely leak the second choice of k. The attacker, given the value for k, can readily recover the user's private key x. This setup attack is rather odd since the kleptogram is not overtly displayed. Instead, it is recovered during the signature verification process.

The presence of the mechanism cannot be detected in a tamper-resistant black-box implementation (i.e., Capstone which is a key escrow technology which also employs the DSA system), for the same reasons as in the ElGamal digital signature setup. Note that this kleptographic attack assumes that the device was implemented with a priori knowledge of the values for g, p, and q that the user will use. One can envision a scenario in which the NIST invites several corporations to agree on a choice of parameters using the NIST designed prime number generation method reiterated in [Schneier]. With this setup attack, "trapdoor primes" (originally suspected in DSA) are not needed, any primes will do.

Theorem 3 *DSA has a regular setup version.*

If we trust that the device is indeed tamper-proof, we can leak a private key over two signatures while at the same time letting the user choose his own values for p, q, and g. This can be done by including the attacker's private key X (say, 511 bits in length) within the device. The attack is clearly not a setup attack (it includes the attacker's private key!), but leaks keys at a very high bandwidth, and is very flexible. This attack obviously relies heavily on the tamper-proof nature of the device in question.

4.1 Generalized Information Leakage in DSA

We have shown how the private key x can be leaked over two DSA signatures. Clearly, we can then use the Simmons channel and leak a message mod q in a third signature by setting k equal to that message. But, we can do better. In this section we will show how to leak a message mod q of our choosing over two signatures, in addition to the private key x. But, before doing so we will point out two weaknesses in the Simmons channel. Suppose Alice wants to send Bob the subliminal message "160 bit long string." on two separate occasions. Since the string is 160 bits long, it will occupy the entire value k (assuming we send ASCII text). Hence, the two values for r, where $r = (g^k \ mod \ p) \ mod \ q$ will be identical, and are easily noticed by the warden. Thus, we are forced to break down the message in order to introduce randomness. But then we need more bandwidth to send the message. Also note that without introducing randomness, the warden can mount guessed plaintext attacks. The warden simply guesses that Alice will send the string k = "160 bit long string." and then verifies that $r = (g^k \ mod \ p) \ mod \ q$.

The method we will now describe accomplishes this generalized (1,2)-leakage and avoids these drawbacks. The method for accomplishing this is subtle, and uses a "feed-back" like algorithm. Suppose that Alice is in prison and wants to send a 160 bit message $m < q$ to Bob, who is on the outside. To do so, Alice takes the first message M_1 to be signed, and computes g_1 based on M_1 where g_1 is an element in Z_p with order q. To compute g_1, Alice finds the smallest value $w > 0$ such that $H^w(M_1) \ mod \ p$ generates Z_p. Alice then sets $g_1 = H^w(M_1)^{(p-1)/q} \ mod \ p$. Alice computes the signatures (r_1, s_1) and (r_2, s_2) using her private key x as follows:

1. $k_1 = ((g_1{}^x \ mod \ p) \ mod \ q)m \ mod \ q$
2. $r_1 = (g^{k_1} \ mod \ p) \ mod \ q$
3. $s_1 = k_1{}^{-1}(H(M_1) + xr_1) \ mod \ q$
4. Calculate k_2 using the regular setup in DSA
5. $r_2 = (g^{k_2} \ mod \ p) \ mod \ q$
6. $s_2 = k_2{}^{-1}(H(M_2) + xr_2) \ mod \ q$

Upon receiving the two signatures, Bob can recover m as follows:

1. Bob recovers x using the regular setup in DSA

2. $k_1 = {s_1}^{-1}(H(M_1) + xr_1) \ mod \ q$
3. Bob computes g_1 using M_1 in the same way that Alice did
4. $m = k_1(({g_1}^x \ mod \ p) \ mod \ q)^{-1} \ mod \ q$

In the above algorithm, Alice sends the subliminal message m and a kleptogram in the first signature. The kleptogram is then used to securely compromise the second signature. Bob then recovers k_2, and thus x from the second signature. Bob then takes x and goes back to the first signature, and recovers k_1. Using k_1 Bob recovers the message m. So, Bob takes x from the second signature and "feeds it back" into the first signature to reveal the subliminal message.

Note that x is a shared secret between Alice and Bob. Thus $({g_1}^x \ mod \ p) \ mod \ q$ is a shared secret between Alice and Bob. It is this secret that is used to blind the subliminal message m. The attack is therefore not subject to guessed plaintext attacks, since the warden must guess $({g_1}^x \ mod \ p) \ mod \ q$, in addition to m. Also, since there is a one-to-one mapping between the shared secrets $({g_1}^x \ mod \ p) \ mod \ q$ and the messages M_1 being signed, and since there is no need to sign the same message twice, Alice can send the same subliminal message twice and the values for r will be different.

This whole attack has the drawback that the device will always choose the same k for a given message M being signed. So, when designing black-box devices like Capstone, we might want to randomize some of the upper order bits of m so that it will *look like* the device is really choosing k randomly (joke). So, not only is it possible to leak DSA private keys over two DSA signatures, but it is also possible for devices to leak 160 bits of the devices own choosing at the same time. This channel is ideal for leaking symmetric keys chosen by the user.

4.2 Rogue Use of DSA Easily Implies a "Public Key Cryptosystem"

Recall that $g^k \ mod \ p$ can be recovered from (r, s) by computing the expression ${g^{s^{-1}H(M)}y^{s^{-1}r}} \ mod \ p$. So, Alice can send a DSA signed message to Bob that is effectively public key encrypted as follows. Alice chooses k randomly and raises Bob's DSA public key y to this k, thereby yielding a secret Diffie-Hellman key $z \ mod \ p$. Alice then encrypts this message using z in a symmetric cipher. Alice signs the resulting ciphertext file using her DSA private key and k. Using (r, s), Bob can recover $g^k \ mod \ p$. By raising $g^k \ mod \ p$ to his private key, Bob recovers z. The only information that is sent is the encrypted file (which is 'plaintext') and the signature (r, s). Note that z will be different each time a message is sent, because k will be different.

In [NR94] it was shown "How to Securely Integrate the DSA to Key Distribution" by sending the pair (r, s) with $H(M) = 1$. Here we are not fixing $H(M)$ which would have been quite noticeable. We are in fact *spoofing* normal DSA signed messages to send (effectively) public key encrypted signed messages at the same time. Thus we have shown that:

Theorem 4 *A DSA message/signature pair $(m, (r, s))$ signed by Alice and sent to Bob can be abused to be a pair consisting of a probabilistic public key encrypted message m' encrypted for Bob and signed by Alice.*

Note that Alice and Bob can establish the secret key z, thus:

Theorem 5 *A DSA message signature pair $(m,(r,s))$ signed by Alice and sent to Bob can be abused to be a key exchange message establishing a secret key between Alice and Bob.*

Let us recall that one of the criticisms of the DSA was that DSA does not provide for secret key distribution. In response, [SB92] stated *"The DSA does not provide for secret key distribution because DSA is not intended for secret key distribution"*. Yet we have shown that DSA can be used essentially and quite directly as a PKCS.

4.3 Device Marking

Note that the DSA setup requires that only 160 bits of the hidden field element (which is at least 512 bits) be used for k in the subsequent signature. The remaining 352 bits can be used to compromise other algorithms in a black-box implementation. Furthermore, note that since the user's private key x can be securely derived, the device can also leak information securely using k. This makes for the following rather inviting facility.

Each device could have a unique 26-bit serial number. The device could take 160 bits of the hidden field element, and use 134 of them as the lower order bits of the subsequent k. The other 26 bits can be XORed with the serial number. The result can be used to form the upper order bits of the subsequent k. Note that the attacker must now try 2^{26} possibilities to derive the correct k. However, once the user's private key is recovered, along with the 26 bit pad, the attacker knows exactly which device was used to compute the signature. This allows the device to securely and subliminally mark signatures that it outputs. This marking is essentially the signature of the device embedded within the signature of the user. If users primarily use their own devices to sign their own documents, then this mechanism can both help detect forgeries if x becomes known to an adversary and can be used to, for example, find thieves (who steal the device itself).

5 Regular Setup in the Schnorr Digital Signature Scheme

The following is a quick overview of Schnorr [Sc91]. Let p and q be primes such that q divides $p-1$. Let g be a number such that $g^q = 1 \ mod \ p$. Let $s < q$ be the randomly chosen private key. The public key is $v = g^{-s} \ mod \ p$.

1. Alice picks $r < q$ randomly and computes $x = g^r \ mod \ p$
2. Alice computes $e = H(m,x)$ and sets $y = r + se \ mod \ q$
3. the signature of m is (e,y)

Here H is a one-way hash function. To verify the signature, Bob computes $z = g^y v^e \ mod \ p$ and then makes sure that $e = H(m,z)$. Note that $z = x = g^r \ mod \ p$. Thus, for a valid signature, z can be used to leak a hidden field element. In this respect, the (1,2)-leakage setup in Schnorr is very similar to the setup in DSA.

Theorem 6 *The Schnorr signature algorithm has a regular setup version.*

6 Setup Attacks on Elliptic Curve Cryptosystems

The discrete log setup extends directly to elliptic curve cryptosystems. Let E be an elliptic curve defined over F_q and let B be a publicly known point on E. The attacker chooses a random integer x of order of magnitude q, which he keeps private. The attacker includes in the cryptosystem the point $xB \in E$. B is analogous to g and xB is analogous to y in the discrete log attack. The attack proceeds in exactly the same way as described before, except that we calculate a pseudo-random point c on E as opposed to a pseudo-random hidden field element in F_p. This point is hashed in order to determine the subsequent value k to be used.

Note that a complication arises in trying to calculate the subsequent value k to be used in the cryptosystem. Let $\#E$ denote the number of points on E. We need a value k uniformly distributed in $[0..\#E-1]$. Hasse's theorem asserts that $q+1-2\sqrt{q} \leq \#E \leq q+1+2\sqrt{q}$. Even if say, $q = \#E$, we could not simply set k to be the left coordinate of c since there could be many pairs (x, y) not on the curve with $x < q$. We cannot simply set k to be the right coordinate of c since there could be points (x, y) and (x', y) where $x \neq x'$ and $y < q$. It is well known that there is no convenient method known to deterministically generate points on E. Hence, finding a function that calculates an unbiased k given c is a difficult problem. One possible solution is to use a hash function as a random oracle to hash c to a value between 0 and $\#E - 1$. An elegant solution to the bias problem in elliptic curve setups is left as an open problem. We close this section by noting that the Menezes-Vanstone PKCS [MV93] can have a setup using this method.

Theorem 7 *Menezes-Vanstone PKCS has a regular setup version.*

7 SETUPs in key exchanges

We now describe how to employ the methodology in the authenticated key exchange protocols given in [Stin95]. The following is a review of MTI. Let g be a primitive element modulo the prime p. Each user U has an ID string, ID(U), a secret exponent a_u $(0 \leq a_u \leq p-2)$, and a corresponding public value $b_u = g^{a_u} \ mod \ p$. The TA has a signature scheme with a verification algorithm V_{TA} and a secret signing algorithm S_{TA}. Each user U will have a certificate $C(U) = (ID(U), b_u, S_{TA}(ID(U), b_u))$. To exchange keys, users U and V do the following.

1. U chooses r_u at random, $0 \leq r_u \leq p-2$ and computes $s_u = g^{r_u} \ mod \ p$
2. U sends $(C(U), s_u)$ to V
3. V chooses r_v at random, $0 \leq r_v \leq p-2$ and computes $s_v = g^{r_v} \ mod \ p$
4. V sends $(C(V), s_v)$ to U

5. U computes $K = {s_v}^{a_u} {b_v}^{r_u} \ mod\ p$, where b_v is obtained from $C(V)$. V computes $K = {s_u}^{a_v} {b_u}^{r_v} \ mod\ p$, where b_u is obtained from $C(U)$.

Note that in the first exchange, r_u and r_v for the second exchange can be leaked if the devices belonging to *both* U and V have a setup. The attacker then knows the K of the second round since $K = {b_v}^{r_u} {b_u}^{r_v} \ mod\ p$. Finding the setup in the Girault Key Agreement Protocol is left as an exercise for the reader.

8 Conclusion

We have demonstrated the prevalence of kleptographic attacks and the applicability of the kleptographic point of view. We presented a direct methodology for the systematic search for attacks based on kleptographic relations. The discrete log kleptogram was used in particular to implement setups in numerous systems, and influenced potential abuses of the DSA signature scheme.

References

[DSS91] Proposed Federal Information Processing Standard for Digital Signature Standard (DSS). In v. 56, n. 169 of *Federal Register*, pages 42980–42982, 1991.

[ElG85] T. ElGamal. A Public-Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. In *Advances in Cryptology—CRYPTO '84*, pages 10–18, 1985. Springer-Verlag.

[GM84] S. Goldwasser, S. Micali. Probabilistic Encryption. *J. Comp. Sys. Sci.* 28, pp 270-299, 1984.

[MV93] A. Menezes, S. Vanstone. Elliptic curve cryptosystems and their implementation. In *Journal of Cryptology*, volume 6, pages 209–224, 1993.

[NR94] K. Nyberg, R. Rueppel. Message Recovery for Signature Schemes Based on the Discrete Logarithm Problem. In *Advances in Cryptology—EUROCRYPT '94*, pages 182–193, 1994. Springer-Verlag.

[RSA78] R. Rivest, A. Shamir, L. Adleman. A method for obtaining Digital Signatures and Public-Key Cryptosystems. In *Communications of the ACM*, volume 21, n. 2, pages 120–126, 1978.

[SB92] M. Smid, D. Branstad. Response to Comments on the NIST Proposed Digital Signature Standard. In *Advances in Cryptology—CRYPTO '92*, pages 76–88, 1992. Springer-Verlag.

[Sc91] C. Schnorr. Efficient signature generation by smart cards. In *Journal of Cryptology*, volume 4, pages 161–174, 1991.

[Schneier] B. Schneier. Applied Cryptography, pages 309–310, 1994. John Wiley and Sons, Inc.

[Sim85] G. J. Simmons. The Subliminal Channel and Digital Signatures. In *Advances in Cryptology—EUROCRYPT '84*, pages 51–57, 1985. Springer-Verlag.

[Sim93] G. J. Simmons. Subliminal Communication Is Easy Using the DSA. In *Advances in Cryptology—EUROCRYPT '93*, 1993. Springer-Verlag.

[Stin95] D. R. Stinson. Cryptography: theory and applications, 1995, CRC Press.

[YY96] A. Young, M. Yung. The Dark Side of Black-Box Cryptography. In *Advances in Cryptology—CRYPTO '96*, pages 89–103, Springer-Verlag.

[YY97] A. Young, M. Yung. Kleptography: Using Cryptography against Cryptography. In *Advances in Cryptology—EUROCRYPT '97*, pages 62–74, 1997. Springer-Verlag.