



Kleptography: Using Cryptography Against Cryptography

Adam Young* and Moti Yung**

Abstract. The notion of a Secretly Embedded Trapdoor with Universal Protection (SETUP) has been recently introduced. In this paper we extend the study of stealing information securely and subliminally from black-box cryptosystems. The SETUP mechanisms presented here, in contrast with previous ones, leak secret key information **without** using an explicit subliminal channel. This extends this area of threats, which we call “kleptography”.

We introduce new definitions of SETUP attacks (strong, regular, and weak SETUPS) and the notion of m out of n leakage bandwidth. We show a strong attack which is based on the discrete logarithm problem. We then show how to use this setup to compromise the Diffie-Hellman key exchange protocol. We also strengthen the previous SETUP against RSA. The strong attacks employ the discrete logarithm as a one-way function (assuring what is called “forward secrecy”), public-key cryptography, and a technique which we call probabilistic bias removal.

Key words: cryptanalytic attacks, kleptography, leakage bandwidth, Discrete Log, Diffie-Hellman, RSA, design and manufacturing of cryptographic devices and software, black-box devices, subliminal channels, information hiding, SETUP mechanisms, randomness, pseudorandomness.

1 Introduction

Numerous problems and subtleties exist when constructing a cryptosystem for use, since designing and manufacturing secure cryptosystems is a demanding task. Some of these problems are immediate and known, yet they require diligent engineering. Other issues are more involved or are yet unknown.

One area where problems have been recognized is in the information-hiding aspect of cryptosystems, and in particular the existence of “subliminal channels” in cryptosystems. Subliminal channels can be used to convey information in the output of a cryptosystem in a way that is different from the intended output. This notion was put forth by Simmons [Sim85, Sim94]. Other works on subliminal channels are [Des90] which showed an RSA channel and [KL95] which showed how to hide a shadow public key inside a key distribution method. The usage of subliminal channels expose information universally (to anyone).

* Dept. of Computer Science, Columbia University Email: ayoung@cs.columbia.edu.

** CertCo, NY, USA. Email: moti@cs.columbia.edu, moti@certco.com

Recently, it was shown that a cryptosystem, when implemented as a black-box (i.e., when the user has only input/output access to the hardware or software cryptographic facility), can be designed such that it gives a unique advantage to the attacker. This is accomplished using SETUP mechanisms [YY96]. SETUPS are unnoticeable in black-box environments and they resist reverse-engineering as well (the device may still use a strong random source).

Indeed, black-box cryptography is both endorsed and employed by the U.S. government (“trusted” hardware devices). It is also in use in the private sector (e.g., embedded cryptography in devices like cellular phones). It is often the case that crucial cryptographic key management functions are implemented in hardware and that companies that produce commercial software implementations of cryptographic systems do not publicize source code to protect proprietary information. Even when specifications are available, users rarely check the validity or compliance of the available implementation against the specifications.

Previously, the SETUP threat employed subliminal channels and combined subliminal channels with public key cryptography (with a private key known only to the attacker). In this paper we present various “kleptographic threats”. Kleptography, in turn, is defined as the “study of stealing information securely and subliminally”; we limit ourselves to the context of cryptographic systems. The kleptographic attacker can steal the secrets securely, and in an exclusive and subliminal manner.

The attack that we present involves public-key cryptography and strong one-way functions, and is in the same spirit as SETUP attacks (avoiding trivial attacks on the pseudorandomness of the device and similar simple attacks where reverse engineering implies knowledge of the future states of the device). What is new in this work is that we show how to implement SETUPS without using explicit subliminal channels. Rather than employing an “information leaking channel,” the implementation, in conjunction with the internal cryptographic tools, *generates* opportunities for leaking information.

In this paper:

1. We refine the notion of a SETUP [YY96] and define the notions of weak, regular, and strong SETUPS.
2. We expand the range of setup attacks that can be carried out on cryptographic devices. We define (m, n) -leakage bandwidth.
3. We present a setup mechanism that employs the discrete logarithm problem; (previously, only weak attacks were known on discrete log systems). We show how it can be embedded within a device that conducts Diffie-Hellman key exchanges.
4. The mechanism is used to strengthen the SETUP in RSA keys (presented in [YY96]), so that after reverse-engineering of the RSA key generation device, one cannot tell whether the past keys that were generated were generated by a kleptographic mechanism or by a regular one.
5. A key technique that is presented is “probabilistic bias removal”. Bias removal simply eliminates the biases of a distribution caused by the algebra employed by the setup mechanism within a cryptographic device.

2 Definitions and Background

A Secretly Embedded Trapdoor with Universal Protection is an algorithm that can be embedded within a cryptosystem to leak encrypted secret key information to the attacker in the output of that cryptosystem [YY96]. This encryption is performed by a PKC function E that is contained within the cryptosystem. E may be a probabilistic public key encryption function [GM84]. The outcome is a strong ‘encryption’ that is leaked in a fashion that is noticeable only to the owner of the private portion of E . The following is the definition of a (regular) setup, which is based on the definition from [YY96].

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.

2.1 Weak SETUP

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).

It may seem that a weak setup is cryptographically insecure. Indeed it is in the sense that it can be detected in poly-time by the owner of the device (but not compromised by anyone). Note however, that the user (owner) must first assume that the device in question contains a SETUP, and must also know exactly how to test the black-box device for the presence of it. Weak SETUPS are sufficient for the case where the end users are in collaboration. An example of this (as shown in [YY96]) is the prisoner’s dilemma of Gus Simmons [Sim85]. In this scenario, Alice is in prison and wants to leave. She contaminates her own

cryptosystem with a weak SETUP so as to leak her private key to Bob through digital signatures. After securely leaking her private key she can send data to him subliminally through digital signatures.

2.2 Strong SETUP

The key aspect of a regular setup is that we assume that the users do not have access to the actual implementation of C . This is in fact necessary for polynomial indistinguishability. Now assume that devices/implementations sometimes use the contaminated algorithm (namely, the setup) and sometimes use the uncontaminated (setup-free) version. Now we can make an interesting strengthening.

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.

A strong setup is a much more powerful notion than a regular setup. To exemplify, consider the following problem. Suppose that we are given a cryptographic device such that with 50% probability it uses the setup mechanism, and with 50% probability it behaves normally (based on a random bit, say). The claim is that if the setup is a strong setup, then a user who is handed the output of such a device cannot tell with probability greater than 50% whether or not the output contains hidden secret key information. The obvious assumption being made here is that the user did not observe the computation (randomness) that yielded the output in question, but otherwise he can observe the device's algorithms and control.

This notion is useful to an attacker as protection against the threat of “key revocation”, since even if the device is reverse-engineered, previously generated setup keys are indistinguishable from normal ones. Furthermore, the decision as to which keys to steal may be dictated by a secret policy used or given as input at the time of stealing. Mathematically, the notion gives an extra challenge beyond the polynomial indistinguishability based on the public key of the attacker and the pseudorandomness which is protected by the device. In fact, the involvement of a hard to invert one-way function and the notion of “forward secrecy” seems to be needed (“forward secrecy” is the notion that applies to key distributions – it requires that compromising the long lived key should not give away previous session keys distributed using the compromised long-lived key). The strong setup further requires that the distributions of the cryptosystem and the setup one be “indistinguishable” – even when given the public keys and tools embedded inside the black-box device. A weak setup in ElGamal signature was presented in [YY96], which is all they achieved based on algebraic properties of the discrete logarithm problem.

2.3 Leakage Bandwidth

We now define the notion of leakage bandwidth in cryptosystems. It defines what can be leaked in cryptographic systems (e.g., key generation or key exchange) that are repetitively invoked.

Definition 4. A (m, n) -leakage scheme is a SETUP mechanism that leaks m keys/secret messages over n keys/messages that are output by the cryptographic device ($m \leq n$).

The discrete log attack that we present is a $(1, 2)$ -leakage scheme where in two key generations we are able to leak one key to the attacker. We will show how this scheme can be extended to become a $(m, m + 1)$ -leakage scheme.

3 Discrete Log based SETUP against Diffie-Hellman

Previously, the underlying strategy was to somehow modify a cryptosystem to ‘display’ the public key encrypted ciphertext of secret key information in the output of the cryptographic device. Such modifications are difficult to come by, since the modification must not interfere with the normal operation of the device, and the SETUPS output must also be embedded in the normal output of the device. Hence, the data that is output by the device is dual in nature. A subliminal channel is the traditional vehicle for leaking such data, since a channel has a known bandwidth and does not interfere with the expected operation of a device. What we will now present is a different approach to leaking data securely.

We will now briefly review the Diffie-Hellman key exchange protocol [DH76]. Alice and Bob want to agree on a secret key using an insecure communication channel. Diffie-Hellman uses the parameters p , which is a large prime, and g which is a generator modulo p . These parameters are public. To establish a secret key k , they do the following. Alice generates a value a randomly, where $a < p - 1$. Bob generates a value b in the same fashion. Alice sends Bob $A = g^a \text{ mod } p$ and Bob sends Alice $B = g^b \text{ mod } p$. They can both compute k , where $k = A^b = B^a \text{ (mod } p)$.

The primary attack that is presented in this paper introduces a setup for Diffie-Hellman. Let p is a large strong prime and g is a generator mod p . The user’s private key is x where x is less than $p - 1$ (as in ElGamal scheme [ElG85]). The user’s public key is (y, g, p) where $y = g^x \text{ mod } p$. To encrypt a message m ($m < p$), k is chosen randomly such that $k < p - 1$. We then compute $r = g^k \text{ mod } p$, and $s = y^k m \text{ mod } p$. The ciphertext of m is the pair (r, s) . To recover m , we compute $s/r^x \text{ mod } p$.

3.1 Discrete Log Attack

Suppose that the only information that we are allowed to display is $g^c \text{ mod } p$ for some $c < p - 1$ (as in Diffie-Hellman). The question is, how can we leak

c efficiently? The following is a way to leak a value, call it c_2 , over the single message $m_1 = g^{c_1} \bmod p$, such that the subsequent message $m_2 = g^{c_2} \bmod p$ is compromised. In this attack we assume that the device is free to choose the exponents used. Let the attacker's 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 hash function. WLOG, assume that H generates values less than $\phi(p)$. The following algorithm describes the operation of the Diffie-Hellman device when it is used two times.

1. For the first usage, $c_1 \in Z_{p-1}$ is chosen uniformly at random
2. The device outputs $m_1 = g^{c_1} \bmod p$.
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. $z = g^{c_1 - W^t Y^{-ac_1 - b}} \bmod p$.
6. $c_2 = H(z)$
7. The device outputs $m_2 = g^{c_2} \bmod p$.

The attacker need only passively tap the communications line, and obtain m_1 and m_2 , in order to calculate c_2 . The value for c_2 is found as follows.

1. $r = m_1^a g^b \bmod p$
2. $z_1 = m_1 / r^X \bmod p$
3. if $m_2 = g^{H(z_1)} \bmod p$ then output $H(z_1)$
4. $z_2 = z_1 / g^W$
5. if $m_2 = g^{H(z_2)} \bmod p$ 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 . The reason for using W will become clear in the next section.

What is strange about the above setup mechanism is that we didn't choose c_2 randomly and then public key encrypt it. Instead, we designated $g^{c_1} \bmod p$ to be the ElGamal encryption of something, and then calculated what that something was. Note that $g^{c_1} \bmod p$ acts as both the first and second parts of the ElGamal encryption of z . So, we are doing ElGamal encryptions (r, s) , where $r = s$. This is made possible due to fact that the device is free to choose its own random parameters. Hence, it is possible to leak an exponent efficiently using exponentiated values $g^c \bmod p$ alone. The discrete log attack, in effect, securely discloses a pseudo-random value c_2 to the attacker and then deliberately uses it in a subsequent message. We call z a *hidden field element* with respect to Y , since it is an element of Z_p that can only be recovered using the trapdoor information in Y (or at least as conjectured). As described, this is a (1,2)-leakage system (note that we never said we could choose our messages explicitly!).

In order for c_2 to be able to take on any value less than $p-1$ we assume that $g_1 = g^{-Xb-W}$, $g_2 = g^{-Xb}$, and $g_3 = g^{1-aX}$ are generators mod p .

Claim 1 z is uniformly distributed in Z_p .

Proof. We have the equation $Y^{ac_1+b}g^{Wt}z = g^{c_1} \bmod p$. Solving for z , we get $z = g^{-Xb-Wt}g^{(1-aX)c_1} = g_i g_3^{c_1} \bmod p$, where i is 1 or 2. But $g_i = g_3^u \bmod p$, for some integer u . So, $z = g_3^{c_1+u} \bmod p$. Since c_1 is chosen uniformly at random, the claim holds. QED.

If H is a pseudo-random function [GGM86], then c_2 can take on any value less than $p-1$ as desired. Note that the attack works when $(p-1)/2$ is a prime (and it also works when it is composite).

3.2 Security of Discrete Log SETUP Mechanism

There are two issues to consider with respect to the discrete log attack. It must be intractable for people other than the attacker to recover c_2 . It must also be intractable for people other than the attacker to detect that this SETUP Mechanism is in use. We consider these in turn.

Claim 2 *The Discrete Log SETUP is secure iff the DH problem is secure.*

Proof. Suppose we have an oracle A that solves the DH problem. So, $A(g^u, g^v) = g^{uv}$. Let $f = g^{c_1}/A(g^{ac_1+b}, Y)$. Clearly, f or f/g^W is z . From z we can readily obtain c_2 . Suppose we have an oracle B that breaks the Discrete Log SETUP mechanism where $B(y, m_1) = (z_1, z_2)$. Here z_1 corresponds to $t = 0$ and z_2 corresponds to $t = 1$. We have g^u and g^v and wish to find g^{uv} . We can use B to solve the DH problem as follows. We run $B(g^v, g^u)$ and take z_1 of the output. We then calculate $f = g^u(g^v)^{-b}z_1$. It follows that $z = f^{1/a} \bmod p$. QED.

We have shown that the setup is secure in the sense that a user, not knowing the random choices of exponents of the device, cannot determine the second exponent c_2 . It remains to show that users cannot detect the presence of the Discrete Log SETUP.

Claim 3 *Assuming H is a pseudorandom function, and that the device design is publicly scrutinizable, the outputs of C and C' are polynomially-indistinguishable.*

Proof. We know that z is uniformly distributed in Z_p from Claim 1. Therefore, since H is a pseudo-random function (whose domain is Z_{p-1}), c_2 is distributed uniformly in Z_{p-1} . So, the exponentiated values that are output by C and C' have polynomially indistinguishable probability distributions. QED.

From Claims 2 and 3 it follows that

Theorem 1 *The Discrete Log problem has a strong setup implementation, assuming DH is hard.*

It remains to explain why the value W was used in the setup mechanism. This is used as a precaution in the case that H were found to be invertible. This precautionary mechanism is intended to further insure undetectability for

black-box implementations. So, suppose that the device is a black-box, the choice of exponents are made available to the user, and H is invertible. Furthermore, suppose that the outcome of t is always zero (i.e., W isn't used). WLOG, assume that $a = 1$ and $b = 0$ is publicly known. The user can detect the presence of the setup probabilistically as follows. The user generates several Diffie-Hellman values, and corresponding exponents. Consider one such pair of exponents c_1 and c_2 . Since H is invertible, the user can calculate z . But, the user does not know Y since the device is a black-box. The user hypothesizes that the attacker's private key X is odd (so g^{1-x} isn't a generator). If this is the case, then the user would expect that if c_1 is even, then g^{c_1}/z would be a residue mod p . If c_1 where odd, then the user would expect that g^{c_1}/z would be a non-residue mod p . Now suppose that the user hypothesizes that X is even. Then if c_1 is odd or even, g^{c_1}/z is always a residue. Hence, under these circumstances, the user can detect the presence of the setup on a probabilistic basis by looking for quadratic residues (or non-residues) modulo p .

3.3 Strong Setup in Diffie-Hellman

The discrete log setup attack can be used to implement a strong setup in Diffie-Hellman, so long as the device does not output the exponents it chooses to the user. Implementing the attack is straightforward. The attacker includes his or her Y within Alice's device. The attacker then need only passively tap the communications line. It is assumed that g and p remain fixed.

Theorem 2 *The Diffie-Hellman key exchange has a $(l, l+1)$ -leakage bandwidth SETUP implementation.*

We need to show that we can increase the bandwidth of the attack dramatically. We can do so by chaining together the values that are leaked. We calculate $c_3 = H(z)$ using the equation $Y^{ac_2+b}g^{W^t}z = g^{c_3} \pmod p$. The value of $g^{c_3} \pmod p$ is then used in the next key exchange. We continue this process, say l times. This permits the leakage l contiguous Diffie-Hellman keys. After l times a new c_1 is chosen entirely random, thus insuring that all such contaminated devices behave differently. Thus, the attack can be expanded to become a $(l, l+1)$ -leakage setup. Note that this attack requires the storage of a small amount of state information to work.

4 Probabilistic Bias Removal Method (PBRM)

Consider the following effective, albeit trivial, setup attack on a hybrid cryptosystem based on RSA and IDEA (PAP is a kleptographic version of PGP where 'Good' is changed to 'Awful' [YY96]). This version contains the attacker's 512 bit RSA public key and requires that its users use 1024 bit public keys. PAP operates as follows. After the user has given PAP his or her own public and private keys, PAP recovers the users prime p , where $n = pq$. PAP then divides p

into two equal length bit-strings and then probabilistically encrypts both using the attacker's public key. The result is two ciphertext bit-strings each of which is 512 bits in length. Since the key size of IDEA is 128 bits, PAP proceeds to leak these bit-strings by using them as the next eight symmetric keys used by the program. This constitutes a (1,8)-leakage setup attack.

If the attacker succeeds in retrieving enough of these session keys (e.g., by convincing the user to e-mail him stuff), then he can compute the user's private key. If the user suspects his PGP is really PAP, then he cannot simply encrypt his prime p and compare, since the encryptions were probabilistic. However, if the user generates enough symmetric keys using PAP he can detect the contamination. The method for doing so was noted by [Sch] in regards to the version of PAP in [YY96]. Note that each of the two ciphertext bit-strings that are leaked are each less than the attacker's public modulus N . The output of the device is therefore biased towards outputting session keys, which when concatenated in sets of four, are less than N , whereas the values should be uniformly distributed in $\{0, 1\}^{512}$. This is in fact a very general problem in kleptography, since it is public key encrypted values that are publicly displayed.

An abstract version of the 'biasing problem' can be stated as follows. We are given a value x that is uniformly distributed in $[1..R]$, and we want a value x' that is uniformly distributed in $[1..S]$, where $R > S/2$. Furthermore, we require that x be easily obtainable from x' . We will now describe our Probabilistic Bias Removal Method (PBRM) which accomplishes this. Assume that we have access to an unbiased coin. We flip the coin and obtain either heads or tails. If $x \leq S - R$ and we get heads then we set $x' = x$. But, if $x \leq S - R$ and we get tails then we set $x' = S - x$. If $x > S - R$ and we get heads, then $x' = x$. If $x > S - R$ and we get tails then we repeat the entire algorithm from the beginning. It is clear that x is readily obtainable from x' , since $x = x'$ unless $x' > R$, in which case $x = S - x'$.

Claim 4 x' is uniformly distributed in S .

Proof. x is chosen uniformly at random from $[1..R]$. So, the probability that a particular x is chosen is $1/R$. In the case that $x \leq S - R$, x' will be set to x with probability $1/2R$ and x' will be set to $S - x$ with probability $1/2R$. Thus the values of x' at the beginning and ending of the range of S are uniformly distributed. It remains to show that the values in the middle have the same probability of occurring. If $x > S - R$, then x' will be set to x with probability $1/2R$. If the toss comes out tails, then the experiment is repeated. QED.

Note that in the version of PAP presented above, if we take $R = Z_N^*$, the values 1, p , and q are not in R . Such minute discrepancies can be ignored however.

5 Strong Setup in RSA Key Generation

In [YY96] a setup for RSA [RSA78] key generation was proposed. This setup constitutes a regular setup but can be modified to be a strong setup. To see why

the previous attack does not constitute a strong setup, consider the following. The user knows his public modulus n , his public exponent e , and his private exponent d . From these he can factor n and recover the secret primes p and q . If the user knows exactly how the attack is implemented (i.e., the attacker's public key, the fixed symmetric key, etc.), then he can detect the mechanism based on p and n . The user simply encrypts p in the same way as the mechanism would and compares the result to the upper order bits of n . If they match, then he has successfully distinguished C' from C in poly-time. However, the setup as stated is a regular setup, since knowledge of the fixed symmetric key is needed to detect any possible bias in the output.

We will now describe a modification to the setup based on the discrete log attack that constitutes a strong setup. This version of PAP contains the attacker's ElGamal public key (Y, g, P) . P is the same size as the prime p being generated. The attacker keeps his private key X secret. Let $a = G(b, c)$ denote a pseudo-random function G that when applied to the data b using the key c produces a value a . Let M be the number of bits in the representation of P . Finally, let K be a fixed symmetric key which need not be secret that is included within the device. Below is the pseudo-code for the setup attack.

1. choose c_1 randomly where $c_1 < P - 1$
2. solve for z in $Y^{ac_1+b}g^{Wt}z = g^{c_1} \bmod P$ (the discrete log attack)
3. remove the bias of z to get z' using the PBRM (assuming that the input of H needs to be distributed uniformly in some domain larger than P), goto step 1 if repeat is necessary
4. set $z'' = H(z')$
5. set lowest order bit of z'' to 1 (so z'' is odd)
6. set $p = z'' + num$ where num is the smallest positive integer that makes p prime (increment in steps of 2 and check odd values for primality. We assume that $num \leq B_1$ where B_1 is some constant)
7. apply PBRM to $g^{c_1} \bmod P$ to get a value v , repeat step 6 as necessary
8. for ($i = 0; i < B_2; i++$) do steps 8 through 12
9. $U = G(v, K + i)$
10. choose the value RND uniformly at random from $\{0, 1\}^M$
11. Let $[U][RND]$ be the concatenation of the bit-strings U and RND
12. solve for q in the equation $[U][RND] = pq + r$
13. if q is prime then set $n = [U][RND] - r$ and goto step 14
14. goto step 1
15. calculate the RSA exponents e and d

To find out if a given public key was created using PAP, the attacker does the following. He first sets U to be the upper order bits of the victim's public modulus n such that there are M bits to the right of this value. He then decrypts U using $K + i$ and where i ranges from 0 to $B_2 - 1$. If any of the resulting values is greater than or equal to N , then a toss of tails occurred in the last application of the PBRM, so the correct value for $g^{c_1} \bmod P$ needs to be calculated. The attacker then decrypts all of the values for $g^{c_1} \bmod P$ using his private key to

get the set of possible values for z . Since the PBRM was used, there are at most two possible values z' for each z . For each z' , we compute $z'' = H(z')$ and set the lowest order bit of z'' to one. We then increment in steps of two to get the set of candidate values for p . Like before, we increment in steps of two to check only odd values. The number of candidate values are limited by the value B_1 . If any of one of the resulting values divides n , then the attacker has successfully factored the victim's modulus. If a factor isn't found, then the attacker decrypts $U + 1$ and proceeds as before. Note that since the PAP ignores the remainder upon dividing $[U][RND]$ by p , it is possible that a borrow bit modified the upper order bits of n . It is for this reason that the attacker must try $U + 1$ as well. If by then, a factor isn't found, the attacker concludes that his version of PAP was not used to generate the public key.

A few explanations for why PAP operates in this way are in order. PAP applies bias removal to $g^{c_1} \bmod P$ to prevent statistical detection of the setup mechanism. We assume that H is a pseudo-random function, so we did not apply bias removal to z'' . Note that the transformation $z'' = H(z')$ insures that p can have a value larger than the attacker's public modulus. The reason for encrypting $g^{c_1} \bmod P$ using G is to take advantage of the pseudo-randomness and to avoid the overhead of excessive modular arithmetic, the amount of which is dictated by the prime number theorem. Hence, this step is essential to ensure a good probability of finding a valid p and q . We implemented the strong setup for RSA. See the appendix for an analysis of its performance.

We would like to briefly add that the setup attacks on DSA and Kerberos given in [YY96] can be readily modified to become strong setups. This can be accomplished by leaking probabilistic public key encrypted data, where the PBRM has been applied to the ciphertext that results from the probabilistic encryption. The probabilistic encryptions prevent the user from detecting the contamination by re-encrypting the secret information (which he knows) and comparing.

5.1 Security of Strong RSA Key Setup

By making certain reasonable cryptographic assumptions, the values for p and q that are chosen by PAP are random.

Lemma 5. *Assuming that p and the upper order bits $[U]$ of $[U][RND]$ are random, q is random in the set of M -bit primes.*

Claim 5 *Assuming the design of PAP is publicly available, the output of C and C' are polynomially indistinguishable.*

Proof. PAP does not make its choices of exponents c_1 known. Hence, Claim 2 applies, and PAP is secure iff the DH problem is hard. Clearly the upper order bits $[U]$ are chosen randomly in PAP. Since p is found from the strong one-way hash (and pseudo-random function [GGM86]) of z' , it follows from lemma 5 that the probability distributions of C and C' are polynomially indistinguishable. QED.

It follows that

Theorem 3 *RSA has a strong setup as long as the DH problem is hard.*

As a side note, this setup can be modified to accommodate the generation of strong primes.

6 Conclusion

We found kleptographic attacks against systems that do not have explicit subliminal channels. The stealing was made more effective by repetitive correlated usage, and by increasing the leakage bandwidth through chaining. It was demonstrated that repeated use of a cryptosystem may generate “implicit channels” for attacks. Chaining, in turn, increases the applicability of stealing via SETUP mechanisms. We also refined and strengthened the notion of SETUP attacks.

Acknowledgments:

We would like to acknowledge Jo Schueth for pointing out the statistical attack on the RSA key setup and Hari Sundaram for improving the efficiency of the PBRM recovery algorithm.

References

- [Des90] Yvo Desmedt. Abuses in Cryptography and How to Fight Them. In *Advances in Cryptology—CRYPTO '88*, pages 375–389, Berlin, 1990. Springer-Verlag.
- [DH76] W. Diffie, M. Hellman. New Directions in Cryptography. In *IEEE Trans. on Information Theory*, 22(6), pages 644–654, 1976.
- [ElG85] T. ElGamal. A Public-Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. In *Advances in Cryptology—CRYPTO '84*, pages 10–18, Berlin, 1985. Springer-Verlag.
- [GGM86] O. Goldreich, S. Goldwasser, and S. Micali, How to Construct Random Functions. *J. of the ACM*, 33(4), pp 210–217, 1986.
- [GM84] S. Goldwasser and S. Micali, Probabilistic Encryption. *J. Comp. Sys. Sci.* 28, pp 270–299, 1984.
- [KL95] J. Kilian and F.T. Leighton. Fair Cryptosystems Revisited. In *Advances in Cryptology—CRYPTO '95*, pages 208–221, Berlin, 1995. 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.
- [Sch] Jo Schueth, public communication (sci.crypt).
- [Sim85] G. J. Simmons. The Subliminal Channel and Digital Signatures. In *Advances in Cryptology—EUROCRYPT '84*, pages 51–57, Berlin, 1985. Springer-Verlag.
- [Sim94] G. J. Simmons. Subliminal Channels: Past and Present. In *European Trans. on Telecommunication*, 5(4), 1994, pages 459–473.
- [YY96] A. Young, M. Yung. The Dark Side of Black-Box Cryptography. In *Advances in Cryptology—CRYPTO '96*, pages 89–103, Springer-Verlag.

A Performance: Strong RSA SETUP

We demonstrated the practicality of the attack by implementing it and noticing that it performs reasonably well (takes longer in general but sometimes it is faster than a comparable setup-free version).

Our program was written in ANSI C and was linked with the GNU MP library version 1.3.2. Our program generates a 512 bit RSA public/private key pair using the strong setup mechanism described in this paper. Our implementation uses truerand of D. Mitchell and M. Blaze as a source of true randomness (it is part of AT&T CryptoLib by J. Lacy, D. Mitchell, W. Schell). These physically random values are used as seeds for a pseudo-random number generator. We chose to use Wheeler and Needham's TEA as our pseudo-random function (any other block cipher like DES will do). We used the probabilistic primality test from Knuth to test the random values. We chose B_1 equal to 256. The value for B_2 was also 256.

Table 1
512 bit RSA key generation times in seconds

Trial	SETUP gen	SETUP decr
1	404	93
2	35	15
3	63	52
4	104	120
5	17	176
6	150	262
7	172	131
8	334	153
9	132	264
10	133	116
Average	154.4	138.2

The SETUP gen column lists the SETUP key generation times. The SETUP decr column lists the amount of time required to derive a private key from the corresponding public key. We note that the times reported may potentially be decreased by doing the following. By simply hashing the pseudorandomly calculated value instead of applying the PBRM and then hashing, it is likely that the key generation times would be shorter. This would of course be done at the expense of not suppling the hash function with inputs that are uniformly distributed. What we see is variability in the timing; it may be possible therefore, to modify a system like PGP to contain a strong RSA SETUP mechanism such that it can't be detected by noticing a "substantial" delay in the key generation times.