A Space Efficient Backdoor in RSA and Its Applications

Adam Young[1] and Moti Yung[2]

[1] LECG LLC*
ayoung@mitre.org
[2] RSA Labs and Columbia University
moti@cs.columbia.edu

Abstract. In this paper we present an RSA backdoor that, for example, can be used for a hardware-based RSA key recovery system. The system is robust in the sense that a successful reverse-engineer is not able to obtain previous nor future RSA private keys that have been/will be generated within the key generation device. The construction employs the notion of two elliptic curves in which one is the "twist" of the other. We present a proof in the random oracle model that the generated RSA key pairs that are produced by the cryptographic black-box are computationally indistinguishable (under ECDDH) from "normal" RSA key pairs, thus ensuring the integrity of the outputs. Furthermore, the security level of the key recovery mechanism is nearly identical to that of the key pair being generated. Thus, the solution provides an "equitable" level of security for the end user. This solution also gives a number of new kleptographic applications.

Keywords: Key recovery, Diffie-Hellman, Decision Diffie-Hellman, SETUP, tamper-resistance, RSA, black-box ciphers, elliptic curve cryptography, twist on elliptic curves.

1 Introduction

The ability to be able to perform recovery is a necessity for large organizations that need timely access to encrypted information assets. Conventional solutions to the problem often involve the use of PKCS #12 files to store private keys for the long-term in encrypted envelopes. For the RSA cryptosystem, it has been shown that the transmission channel that exists in composites can be used to implement a key recovery system that is transparent with respect to the end user [28], thereby elimating the need for numerous PKCS#12 files and similar storage methods.

In this work we present a new and space efficient RSA [24] key recovery system/backdoor in this transparent model that has a running time that is faster than all previous approaches. Recall that a secretly embedded trapdoor with universal protection (SETUP) in RSA key generation utilizes the public key of the designer to "display" an asymmetric ciphertext in a channel[1] in composites

* Author is now at MITRE Corporation.
[1] About $|n|/2$ bits can be displayed in the bit representation of n.

B. Preneel and S. Tavares (Eds.): SAC 2005, LNCS 3897, pp. 128–143, 2006.

[9], thereby allowing an authorized escrow authority to access the RSA private decryption key of the key owner. Furthermore, the reverse-engineer who breaches the black-box and learns its internals is unable to factor the public keys of those who used the key generation black-box.

The primary technical motivation of this work is the following. First, it is a way to assess the consequences of elliptic curve (EC) technology in regards to constructing hardware-based key recovery for RSA and related technologies. The reason why current RSA SETUPs are deficient is the following. The first RSA SETUP [28] in RSA-1024 that was presented in 1996 is no longer secure since it requires that the designer's embedded public key be 512-bits in size. The security parameter of the designer's public key is half that of the RSA key being generated. So, the user's 1024-bit key ends up being no more secure than RSA-512 with respect to the reverse-engineer. This problem results from the fact that a subexponential time algorithm is known that solves the integer factorization problem and this leads to "bulky" embedded RSA ciphertexts (decryptable only by the designer). Recall that in 1996 the 430-bit RSA composite for the RSA-130 factoring challenge was solved [6] while the tougher challenges remained unsolved.[2] In December 2003, the 576-bit RSA composite RSA-576 was factored [23]. So, whereas it was conceivable in 1996 that a 512-bit modulus provided some security, this is certainly not the case now.

What this means is that there is currently no known way to implement a secure SETUP in RSA-1024 key generation. In this paper we solve this practical problem.[3] In fact, the use of a pair of twisted elliptic curves leads to a solution that is so *space efficient* that it can be used to build a SETUP in RSA-768 key generation as well, provided that a sound cryptographic hash function is available.

Another technical motivation relates to run-time efficiency. It has been noted that care must be taken to define the ensemble from which each of the RSA primes p and q is chosen and ensure that the SETUP conforms to this ensemble [27, 8]. An approach to doing so was presented [27] and we follow this approach. However, the expected number of primality tests in [27] is about $O((\log p)^2)$ (due to the Prime Number Theorem). So, a second technical motivation is to develop a recovery system that produces primes drawn from the correct distribution while achieving an expected number of primality tests that is about $O(\log p)$ as in normal key generation. Our technical contributions are as follows:

1. We present the first *strong* SETUP that is secure for RSA keys as small as 768 bits (given the current strengths of factoring and ECC and given a suitable hash function) and that has $O(\log p)$ expected primality tests.
2. We present the first SETUP in RSASSA-PSS that permits a 20-bit message to be securely transmitted within the 160-bit padding field. This is more robust than a straightforward channel [25], since previously transmitted messages remain confidential (they are asymmetrically encrypted) even after reverse-engineering.

[2] At that time RSA Inc. used decimal to title their challenges.

[3] To eliminate any possible confusion: "the problem" is one that the designer faces.

A strong SETUP in RSA key generation [29] permits the key generation "devices" (either hardware or software) to be manufactured identically (there is no need for unique identifier strings). Consider the setting in which some fraction of the deployed RSA key generators have the SETUP in them. A strong SETUP makes the following possible: even if one of the devices with a SETUP is reverse-engineered it is still not possible, given only oracle access, to distinguish the remaining devices that have been SETUP from the "good" ones.[4]

The SETUP is made possible by the use of the elliptic curve Diffie-Hellman (ECDH) key exchange algorithm. To date there is no publicly known subexponential algorithm for solving the elliptic curve discrete logarithm problem (ECDLP) [14]. As a result an ECDH key exchange value is very small, particularly when point-compression is used. This allows us to overcome the bulky ciphertext that results from "displaying" an RSA ciphertext in the channel in RSA composites, thereby allowing us to build a secure SETUP in RSA-1024 key generation. In a nutshell our SETUP carries out a ECDH key exchange between the device and the designer to permit the designer to factor the RSA modulus that is produced. To achieve the indistinguishability requirements of a strong SETUP two elliptic curves are used, one which is a "twist" of the other.

2 Background, Definitions, and Notation

A number of SETUPs in RSA key generation have been presented [28, 29, 27]. Also, approaches have been presented that emphasize speed [8]. This latter approach is intended to work even when Lenstra's composite generation method is used [20] whereas the former three will not. However, all of these approaches fail when half of the bits of the composite are chosen pseudorandomly using a seed [28] (this drives the need for improved public key standards, and forms a major motivation for the present work). It should be noted that [8] does not constitute a SETUP since it assumes that a secret key remains hidden even after reverse-engineering.

We adapt the notion of a strong SETUP [29] to two games. For clarity this definition is tailored after RSA key generation (as opposed to being more general). The threat model involves: a designer, an eavesdropper, and an inquirer.

The designer builds the SETUP into some subset of all of the black-box key generation devices that are deployed. The goal of the designer is to learn the RSA private key of a user who generates a key pair using a device contained in this subset when the designer only has access to the RSA public keys. Before the games start, the eavesdropper and inquirer are given access to the SETUP algorithm in its entirety.[5] However, in the games they play they are not given access to the internals of the particular devices that are used (they cannot reverse-engineer them).

[4] Timing analysis, power analysis, etc. are not considered here, but should be considered in future work. Our goal is to lay the foundation for building a SETUP in RSA keys wherein the security parameter of the user RSA key is at the lower end of being secure.

[5] e.g., found in practice via the costly process of reverse-engineering one of the devices.

Assumptions: It is assumed that the eavesdropper and inquirer are probabilistic poly-time algorithms and that the RSA key generation algorithm is deployed in tamper-proof black-boxes. It is traditional to supply an RSA key generator with 1^k where k is the security parameter (for theoretically meaningful runtimes). However, for simplicity we assume that the generator takes no input and that the security parameter is fixed. It is straightforward to relax this assumption. Let D be a device containing the SETUP mechanism.

Game 1: Let A and B be two key generation devices. A has a SETUP in it and B does not (B is "normal"). One of these is selected uniformly at random and then the inquirer is given oracle access to the selected machine. The inquirer wins if he correctly determines whether or not the selected device has a SETUP in it with probability significantly greater than 1/2.

Property 1: (indistinguishability) The inquirer fails Game 1 with overwhelming probability.

Game 2: The eavesdropper may query D but is only given the public keys that result, not the corresponding private keys. He wins if he can learn one of the corresponding private keys.

Property 2: (confidentiality) The eavesdropper fails Game 2 with overwhelming probability.

Property 3: (completeness) Let (y, x) be a public/private key generated using D. With overwhelming probability the designer computes x on input y.

In a SETUP, the designer uses his or her own private key in conjunction with y to recover x. In practice the designer may learn y by obtaining it from a Certificate Authority.

Property 4: (uniformity) The SETUP is the same in every black-box cryptographic device.

Property 4 implies that there are no unique identifiers in each device. The *importance* of a strong SETUP then, is that it permits the distribution of a compiled binary program in which all of the instances of the "device" will necessarily be identical without diminishing the security of the SETUP. In hardware implementations it simplifies the manufacturing process.

Definition 1. *If an RSA key generation algorithm satisfies properties 1, 2, 3, and 4 then it is a* **strong SETUP**.

A method for displaying asymmetric ciphertexts in a fashion that is indistinguishable from random bit strings was put forth in [29]. This is accomplished using the *probabilistic bias removal method* which was also employed in [1]. Other recent work in this area includes [21].

The present work utilizes the notion of a "twist" on an elliptic curve over $\mathbb{F}_q$. For typical elliptic curves used in cryptography (e.g., the curves in FIPS 186-2) only about half of $\mathbb{F}_q$ corresponds to x-coordinates on a given curve. However, by

utilizing two curves—one which is a twist of the other, it is possible to implement a trapdoor one-way permutation from $\mathbb{F}_q$ onto itself. The notion of a twist has been used to implement these types of trapdoor one-way permutations which have the (conjectured) property that inversion is exponential in the security parameter [19]. For the RSA permutation inversion is subexponential in the security parameter.

The notion of a twist has also been used to implement strong pseudorandom bit generators and to achieve a simple embedding of plaintexts in the EC version [17, 18] of ElGamal [12]. Recently, twists have been shown to be useful in the problem of implementing a PKCS in which the ciphertexts appear to be uniformly distributed bit strings [21]. In a related fashion, we use the notion of a twist to produce Diffie-Hellman (DH) [10] key exchange values that appear to be uniformly distributed bit strings.

The following is some notation that is used. Let $A \bigoplus B$ denote the bitwise exclusive-or of bit string A with bit string B where $|A| = |B|$. Let $x \approx y$ denote that the integer x is approximately equal to y. Let $x \in_R S$ denote the selection of an element x uniformly at random from set S. Uppercase is used to denote a point on an elliptic curve and lowercase is used to denote the multiplicand. So, $P = kG$ denotes the EC point P that results from adding the point G to itself k times. Let $\#E_{a,b}(\mathbb{F}_q)$ denote the number of points on the elliptic curve $E_{a,b}$ that is defined over $\mathbb{F}_q$. In the pseudocode that is provided, logical indentation will be used to show flow-control (e.g., the body of an "if" statement is indented to the right).

3 System Setup

The key generation algorithm utilizes a pair of binary curves. Each curve is described by the Weierstrass equation $E_{a,b}$ given by $y^2 + xy = x^3 + ax^2 + b$. Here the coefficients a and b are in $\mathbb{F}_{2^m}$ and $b \neq 0$. We use the standard group operations for binary elliptic curve cryptosystems. The value m should be an odd prime to avoid the possibility of the GHS attack [13]. Also, these curves must provide a suitable setting for the elliptic curve decision Diffie-Hellman problem (ECDDH). We mention this since for certain elliptic curves, DDH is tractable [16].

It is well known that Hasse's inequality implies that $|\#E_{a,b}(\mathbb{F}_{2^m}) - 2^m - 1| < 2 * 2^{m/2}$. Recall that if the trace $\mathrm{Tr}_{\mathbb{F}_{2^m}/\mathbb{F}_2}(a) \neq \mathrm{Tr}_{\mathbb{F}_{2^m}/\mathbb{F}_2}(a')$ then $E_{a,b}$ and $E_{a',b}$ are "twists" of one another. When two such curves are twists of one another then for every $x \in \mathbb{F}_{2^m}$ there exists a $y \in \mathbb{F}_{2^m}$ such that (x, y) is a point on $E_{a,b}$ or $E_{a',b}$. The two possibilities are as follows. Either (x, y) and $(x, x + y)$ are points on the same curve or $(x, y) = (0, \sqrt{b})$ is on both curves.

The sum of the number of points on both curves is given by $\#E_{a,b}(\mathbb{F}_{2^m}) + \#E_{a',b}(\mathbb{F}_{2^m})$ which is equal to $2^{m+1} + 2$. It follows from Hasse's inequality that $\#E_{a,b}(\mathbb{F}_{2^m}) \approx \#E_{a',b}(\mathbb{F}_{2^m}) \approx 2^m$.

Since m is odd $\mathrm{Tr}_{\mathbb{F}_{2^m}/\mathbb{F}_2}(0) = 0$ and $\mathrm{Tr}_{\mathbb{F}_{2^m}/\mathbb{F}_2}(1) = 1$. So, we use $E_{0,b}$ and $E_{1,b}$ as a pair of twisted curves. It remains to choose suitable curves that resist known cryptanalytic attacks (e.g., satisfying the MOV condition). Using point-counting techniques it is known how to efficiently generate two curves $E_{0,b}$ and

$E_{1,b}$ with orders $4q_0$ and $2q_1$, respectively, where q_0 and q_1 are prime. $E_{0,b}$ will have a cofactor of 4 and $E_{1,b}$ will have a cofactor of 2.

Once two such curves are found, a base point G_0 having order q_0 that is on $E_{0,b}(\mathbb{F}_{2^m})$ is found as well as a base point G_1 having order q_1 that is on $E_{1,b}(\mathbb{F}_{2^m})$. Using these base points, the designer generates the EC private key $x_0 \in_R \{1, 2, ..., q_0 - 1\}$ and corresponding public key $Y_0 = x_0 G_0$. The designer also generates $x_1 \in_R \{1, 2, .., q_1 - 1\}$ and corresponding public key $Y_1 = x_1 G_1$. The values (G_0, G_1, Y_0, Y_1) are included in the RSA key generation device.

4 Building Blocks

By using point compression, the SETUP is able to make efficient use of space. The ECC point will be embedded in the upper order bits of the RSA modulus that is being SETUP using a well known channel in composites (see [28]). A point (x, y) on the binary curve over $\mathbb{F}_{2^m}$ can be uniquely expressed using $m+1$ bits [26]. The compressed point is $(x, ybit)$ where $ybit \in \{0, 1\}$.

We define $\mathsf{PointCompress}(E_{v,b}, P)$ to be a function that compresses the point $P = (x, y)$ on curve $E_{v,b}$ and outputs $(x \mid\mid ybit)$ which is the compressed representation of (x, y). Also, we define $\mathsf{PointDecompress}(E_{v,b}, x \mid\mid ybit)$ to be a function that decompresses $(x \mid\mid ybit)$ and outputs (P, w). If $w = 1$ then P is the decompressed point on the curve $E_{v,b}$. If $w = 0$ then $(x, ybit)$ is not a point on the curve $E_{v,b}$ and P is undefined.

The following algorithm is used to generate the DH key exchange parameter and the DH shared secret. The algorithm effectively conducts an ECDH key exchange between the device and the designer wherein: the shared secret is used to generate one of the RSA primes, and the public DH paramter is *displayed* in the upper order bits of the published RSA modulus. The function below returns the public and private strings that the device uses for this "key exchange."

GenDHParamAndDHSecret():
Input: none
Output: $s_{pub}, s_{priv} \in \{0, 1\}^{m+1}$
1. with probability $\frac{4q_0 - 1}{2^{m+1}}$ set $a = 0$ and with probability $\frac{2q_1 - 1}{2^{m+1}}$ set $a = 1$
2. choose k uniformly at random such that $0 < k < q_a$
3. choose $\mu \in_R \{0, 1, 2, 3\}$
4. set $P = kG_a$
5. solve for Q such that $P = 2Q$ and Q has order $2q_a$
6. if $a = 0$ then
7. if $\mu = 1$ then set $P = Q$
8. if $\mu \in \{2, 3\}$ then choose Q_1 randomly such that $Q = 2Q_1$ and set $P = Q_1$
9. if $a = 1$ then
10. if $\mu \in \{2, 3\}$ then set $P = Q$
11. set $s_{pub} = \mathsf{PointCompress}(E_{a,b}, P)$
12. set $s_{priv} = \mathsf{PointCompress}(E_{a,b}, kY_a)$ and return (s_{pub}, s_{priv})

The "public" DH key exchange value is s_{pub}. The shared secret is s_{priv}. The following is used to recover the shared secret.

RecoverDHSecret(s_{pub}, x_0, x_1):
Input: $s_{pub} \in \{0,1\}^{m+1}$ and EC private keys x_0, x_1
Output: $s_{priv} \in \{0,1\}^{m+1}$
1. set $v = 0$
2. $(P_1, w) =$ PointDecompress$(E_{0,b}, s_{pub})$
3. if $(w = 0)$ then
4. compute $(P_1, w) =$ PointDecompress$(E_{1,b}, s_{pub})$
5. set $v = 1$
6. set $P_1 = 2^i P_1$ where $i \in \{0,1,2\}$ is the smallest value making P_1 have prime order
7. compute $P_2 = x_v P_1$
8. return $s_{priv} =$ PointCompress$(E_{v,b}, P_2)$

Let Π_θ be the set of all permutations from $\{0,1\}^\theta$ onto itself. We assume that θ is even. The SETUP utilizes the family of permutations $\pi_\theta : \{0,1\}^\theta \rightarrow \{0,1\}^\theta$ for $i = 1, 2, 3, ...$ where π_θ is chosen randomly from Π_θ. We assume that π_θ and π_θ^{-1} are efficiently computable *public* functions (e.g., they are oracles). Given a random oracle $\mathcal{H}$ this family can be constructed. (We assume the Random Oracle model).

In practice this family can be heuristically implemented using strong cryptographic hash functions. For instance, let $H : \{0,1\}^* \rightarrow \{0,1\}^{160}$ and $F : \{0,1\}^* \rightarrow \{0,1\}^{160}$ be distinct cryptographic hash functions.

$\pi_{320}(x)$:
Input: $x \in \{0,1\}^{320}$
Output: $y \in \{0,1\}^{320}$
1. let x_u and x_ℓ be bit strings such that $x = x_u \mid\mid x_\ell$ and $|x_u| = 160$
2. return $y = (x_u \bigoplus F(x_\ell \bigoplus H(x_u))) \mid\mid (x_\ell \bigoplus H(x_u))$

$\pi_{320}^{-1}(x)$:
Input: $y \in \{0,1\}^{320}$
Output: $x \in \{0,1\}^{320}$
1. let y_u and y_ℓ be bit strings such that $y = y_u \mid\mid y_\ell$ and $|y_u| = 160$
2. return $x = (y_u \bigoplus F(y_\ell)) \mid\mid (y_\ell \bigoplus H(y_u \bigoplus F(y_\ell)))$

Note that the transformations $(\pi_\theta, \pi_\theta^{-1})$ are similar to the padding scheme used in RSA-OAEP [3].

The following defines the "public" specification of allowable RSA primes. This definition requires that each of the two most significant bits be set to 1.

IsAcceptablePrime(e, len, p_1):
Input: RSA exponent e, required bit length len of p_1, candidate prime p_1
Output: true iff p_1 is an acceptable prime, false otherwise
1. if $(|p_1| \neq len)$ then halt with false
2. if the two uppermost bits of p_1 are not 11_2 then halt with false

3. if p_1 is composite then halt with false
4. if $(\gcd(p_1 - 1, e) \neq 1)$ return false else return true

Let $\mathcal{R} : \{0,1\}^* \rightarrow \{0,1\}^\infty$ be a random oracle as defined in [2]. The function $\mathsf{GetBlock}(str, i, \ell)$ returns ℓ consecutive bits of bit string str. The first such bit is the bit at bit position i of str. The bits are ordered from left to right starting with 0. For example, if $str = \mathcal{R}(s) = 01101001...$ then $\mathsf{GetBlock}(str, 0, 4) = 0110$, $\mathsf{GetBlock}(str, 1, 4) = 1101$, and $\mathsf{GetBlock}(str, 4, 4) = 1001$.

$\mathsf{GenPrimeWithOracle}(s, len, e)$:
Input: $s \in \{0,1\}^{m+1}$, required bit length len, RSA exponent e.
Output: Acceptable RSA prime p_1 such that $|p_1| = len$
1. set $j = 0$
2. let $u = \mathsf{GetBlock}(\mathcal{R}(s), j * T, T)$
3. choose $\ell \in_R \{0,1\}^{len-T}$
4. let p_1 be the integer corresonding to the bit string $u \mid\mid \ell$
5. if ($\mathsf{IsAcceptablePrime}(e, len, p_1) =$ true) then output p_1 and halt
6. set $j = j + 1$ and goto step 2

The RSA modulus of the user is $n = p_1 q_1$. We require that $|n|/4 \leq T \leq len$. A selection for T is given in Section 5.

5 Application 1: Elliptic Curve SETUP in RSA Key Generation

$N/2$ is the size in bits of each prime in n and e is the RSA exponent (this variable q_1 is different from the elliptic curve value q_1 and should be clear from the context). For simplicity we assume that N is even.

$\mathsf{GetPrimes}_{\mathsf{N,e}}(s_{pub}, s_{priv})$:
Input: $s_{pub}, s_{priv} \in \{0,1\}^{m+1}$
Output: A pair of acceptable RSA primes (p_1, q_1)
1. set $p_1 = \mathsf{GenPrimeWithOracle}(s_{priv}, N/2, e)$
2. choose $s_0 \in_R \{0,1\}^{\theta-(m+1)}$
3. compute $t = \pi_\theta(s_0 \mid\mid s_{pub})$
4. choose $r_2 \in_R \{0,1\}^{N-\theta}$
5. set $n_c = (t \mid\mid r_2)$ /* $\theta + |r_2|$ bits long */
6. solve for (q_1, r_c) in $n_c = q_1 p_1 + r_c$ /* find quotient q_1 */
7. if ($\mathsf{IsAcceptablePrime}(e, N/2, q_1) =$ false) then goto step 2
8. output (p_1, q_1) and halt

Since algorithm $\mathsf{KleptoKeyGen}_{\mathsf{N,e}}$ is deployed in this exact form in all black-box devices, Property 4 of a strong SETUP holds.

$\mathsf{KleptoKeyGen}_{\mathsf{N,e}}()$:
Input: none
Output: A pair of acceptable RSA primes (p_1, q_1)

1. $(s_{pub}, s_{priv}) = \mathsf{GenDHParamAndDHSecret}()$
2. output $(p_1, q_1) = \mathsf{GetPrimes}_{\mathsf{N,e}}(s_{pub}, s_{priv})$ and halt

Once p_1 is found in step 1 it does not change. From the Prime Number Theorem it follows that the expected number of primality tests is $O(\log n)$, the same as in normal RSA key generation. Coppersmith showed that if the $|n|/4$ significant bits of p_1 are known then $n = p_1 q_1$ can be efficiently factored [7]. For typical RSA key generation we can use $T = |n|/4$ in the SETUP. So, we use Coppersmith's method to factor n given the resulting upper order bits.

The value MAX is used to prevent the recovery algorithm from running for too long when it is supplied with an RSA private key that was not generated using the SETUP (e.g., a normal RSA key). By taking $MAX = \lceil \frac{N}{2} * 160 * \ln 2 \rceil$ it follows that if the SETUP was used then the factorization will be found with overwhelming probability.

$\mathsf{KleptoRecoveryKey}_{\mathsf{N,e}}(n, x_0, x_1)$:
Input: the user's RSA modulus n and EC private keys (x_0, x_1)
Output: Factorization of $n = p_1 q_1$ or "failure"
1. let n_s be n represented as a binary string
2. let $t = \mathsf{GetBlock}(n_s, 0, \theta)$
3. let t_0 be the integer corresponding to t
4. for $\beta = 0$ to 1 do:
5. set $t_2 = t_0 + \beta \bmod 2^\theta$
6. let t_3 be the bit string corresponding to t_2
7. set $s_{pub} = \mathsf{GetBlock}(\pi_\theta^{-1}(t_3), \theta - (m+1), m+1)$
8. $s = \mathsf{RecoverDHSecret}(s_{pub}, x_0, x_1)$
9. set $j = 0$
10. let $u = \mathsf{GetBlock}(\mathcal{R}(s), j * T, T)$
11. Attempt to factor $n = p_1 q_1$ by supplying (u, n) to Coppersmith's algorithm [7] and halt with the factorization if it is found
12. set $j = j + 1$
13. if $j < MAX$ then goto step 10
14. output "failure" and halt

The reseason that β is used is to account for a potontial borrow bit being taken from the upper order bits in n_c during the computation of $n = n_c - r_c = q_1 p_1$. A possible configuration of the attack is $N = 768$, $m = 191$, and $\theta = 320$. Observe that $768/2 - 320 = 64$. So, it is not likely that t will have a borrow bit taken from it. It is not hard to see that Property 3 holds. The security of the SETUP is proven in Appendix A.

6 Application 2: Hardware Based Key Escrow

The strong SETUP from Section 5 can be used to implement a lightweight hardware-based key escrow system. The EC private keys (x_0, x_1) are retained by a key recovery authority or may be shared using threshold cryptography among a set of trusted key recovery authorities.

The SETUP is implemented in the hardware devices of the users. The presence and details of the SETUP is publicly disclosed. To recover a given user's RSA private key, the escrow authority or authorities need only obtain the public key of the user to derive the corresponding private key.

7 Application 3: SETUP in RSASSA-PSS

We now present a SETUP in RSASSA-PSS [22] when RSASSA-PSS utilizes SHA-1. Recall that the RSASSA-PSS signature scheme is a Provably Secure Signature (PSS) scheme [4] that constitutes a Signature Scheme with an Appendix (SSA). This scheme is compatible with the IFSSA scheme as amended in the IEEE P1363a draft [15].

For concreteness we set $m = 139$. The use of $\mathbb{F}_{2^{139}}$ is based on the fact that the most recently solved binary EC discrete-logarithm challenge was ECC2-109 [11] and currently one of the easiest open challenges from Certicom is ECC2-131.[6] So, we are admittedly cutting it close.

The encryption and decryption algorithms utilize the cryptographic hash function $H : \{0,1\}^* \rightarrow \{0,1\}^{20}$. The plaintext space is $\{0,1\}^{20}$ and the ciphertext space is $\{0,1\}^{160}$. The asymmetric encryption algorithm $E_{160}(m)$ operates as follows. Let $m \in \{0,1\}^{20}$ be the plaintext message. First, E_{160} computes the value $(s_{pub}, s_{priv}) = \mathsf{GenDHParamAndDHSecret}()$. It then hashes s_{priv} by computing $pad = H(s_{priv})$. The asymmetric ciphertext is $c_1 = (s_{pub} \mid\mid (pad \bigoplus m))$.

The decryption algorithm $D_{160}(c_1)$ operates as follows. It extracts s_{pub} from the asymmetric ciphertext c_1. Algorithm D_{160} then computes the value $s_{priv} = \mathsf{RecoverDHSecret}(s_{pub}, x_0, x_1)$. It then computes $pad = H(s_{priv})$. This pad is then XORed with the 20 least significant bits of c_1 to reveal m.

The following is the SETUP in RSASSA-PSS. The signing algorithm can be used to transmit any 20 bits of information m (e.g., bits of the RSA private key, bits of a symmetric key, etc.) through RSASSA-PSS. It does so by computing $c_1 = E_{160}(m)$ and using c_1 as the random 160-bit "salt" in RSASSA-PSS. The salt/ciphertext is pseudorandom and can be recovered by anyone that is able to perform digital signature verification. However, only the designer who knows (x_0, x_1) can decrypt the salt and recover m.

Note that (E_{160}, D_{160}) is malleable and so does not achieve rigorous notions of security for a PKCS. To see this note that an active adversary can flip plaintext bit i where $0 \leq i \leq 19$ by XORing "1" with the corresponding ciphertext bit.

However, for many applications this asymmetric cryptosystem may provide sufficient security. In the SETUP in RSASSA-PSS, an active adversary that changes a bit as such will with overwhelming probability invalidate the signature. So, in this application of E_{160} non-malleability is achieved.

This SETUP differs in a fundamental way from most channels since confidentiality of m holds even if the cryptographic black-box is opened and scrutinized. Also, the approach of [21] cannot be used to implement this since it involves a

[6] There is an open Koblitz curve challenge called ECC2K-130 as well.

hash field. This hash makes security against chosen ciphertext attacks possible, but causes the minimum-length ciphertext to exceed 160 bits.

This SETUP applies equally well to the padding field in RSA-OAEP. In that scenario the designer must solicit an encrypted message from the user (since in general OAEP padding is not publicly obtainable). In this scenario, when a message is signed and encrypted using PKCS #1, it is possible to transmit $20 + 20 = 40$ bits in a SETUP. This is enough transmit a 64-bit key used by the user to secure other communications. Also, if the channel is repeated a constant number of times many cryptographic secrets can be transmitted (while being protected against reverse-engineering).

8 Conclusion

We presented a key recovery system for factoring based cryptosystems that uses elliptic curve technology. Specifically, we updated SETUP algorithms for use with RSA-1024. The SETUP achieves the notion of a strong SETUP and employs the notion of twisted elliptic curves in a fundamental way. Finally, we showed a SETUP in RSASSA-PSS and pointed out that the RSA digital signatures that result have the added advantage of providing non-malleability of the SETUP ciphertexts.

References

1. L. von Ahn, N. J. Hopper. Public-Key Steganography. In *Advances in Cryptology—Eurocrypt '04*, pages 323–341, 2004.
2. M. Bellare, P. Rogaway. Random Random oracles are practical: A paradigm for designing efficient protocols. In *1st Annual ACM CCCS*, pages 62–73, 1993.
3. M. Bellare, P. Rogaway. Optimal Asymmetric Encryption. In *Advances in Cryptology—Eurocrypt '94*, pages 92–111, 1995.
4. M. Bellare and P. Rogaway. PSS: Provably Secure Encoding Method for Digital Signatures. Submission to IEEE P1363 working group, August 1998.
5. D. Boneh. The Decision Diffie-Hellman Problem. In *Third Algorithmic Number Theory Symposium*, LNCS 1423, pages 48–63, 1998.
6. J. Cowie, B. Dodson, R.M. Elkenbracht-Huizing, A. K. Lenstra, P. L. Montgomery, J. Zayer. A world wide number field sieve factoring record: On to 512 bits. In *Advances in Cryptology—Asiacrypt '96*, pages 382–394, 1996.
7. D. Coppersmith. Finding a small root of a bivariate integer equation; factoring with high bits known. In *Eurocrypt '96*, pages 178–189, 1996.
8. C. Crépeau, A. Slakmon. Simple Backdoors for RSA Key Generation. In *The Cryptographers' Track at the RSA Conference*, pages 403–416, 2003.
9. Y. Desmedt. Abuses in Cryptography and How to Fight Them. In *Advances in Cryptology—Crypto '88*, pages 375–389, 1988.
10. W. Diffie, M. Hellman. New Directions in Cryptography. In volume IT-22, n. 6 of *IEEE Transactions on Information Theory*, pages 644–654, Nov. 1976.
11. eCompute ECC2-109 Project. ECC2-109 solved April, 2004. Details downloaded from http://www.ecompute.org/ecc2.

12. T. ElGamal. A Public-Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. In *Crypto '84*, pages 10–18, 1985.
13. P. Gaudry, F. Hess, N. Smart. Constructive and Destructive Facets of Weil Descent on Elliptic Curves. In *J. of Cryptology*, v. 15, pages 19–46, 2002.
14. D. Hankerson, A. J. Menezes, S. Vanstone. Guide to Elliptic Curve Cryptography. Preface, Springer-Verlag, Jan. 2004.
15. IEEE P1363 working group. IEEE P1363a D10: Standard Specifications for Public Key Cryptography: Additional Techniques. Nov. 1, 2001.
16. A. Joux, K. Nguyen. Separating DDH from CDH in Cryptographic Groups. In *Journal of Cryptology*, v. 16, n. 4, pages 239–247, 2003.
17. B. S. Kaliski. A Pseudo-Random Bit Generator Based on Elliptic Logarithms. In *Advances in Cryptology—Crypto '86*, pages 84–103, 1986.
18. B. S. Kaliski. Elliptic Curves and Cryptography: A Pseudorandom Bit Generator and Other Tools. PhD Thesis, MIT, Feb. 1988.
19. B. S. Kaliski. One-Way Permutations on Elliptic Curves. In *Journal of Cryptology*, v. 3, n. 3, pages 187–199, 1991.
20. A. K. Lenstra. Generating RSA Moduli with a Predetermined Portion. In *Advances in Cryptology—Asiacrypt '98*, pages 1–10, 1998.
21. B. Möller. A Public-Key Encryption Scheme with Pseudo-Random Ciphertexts. In *ESORICS '04*, pages 335–351, 2004
22. PKCS #1 v2.1: RSA Cryptography Standard. RSA Labs, Jun. 14, 2002.
23. E. Weisstein. RSA-576 Factored. MathWorld Headline News, Dec. 5, 2003. Factored by: J. Franke, T. Kleinjung, P. Montgomery, H. te Riele, F. Bahr and NFSNET (that consisted of D. Lecliar, P. Leyland, R. Wackerbarth).
24. R. Rivest, A. Shamir, L. Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. CACM, v. 21, n. 2, pages 120–126, Feb. 1978.
25. G. J. Simmons. Subliminal Channels: past and present. In *European Trans. on Telecommunications*, v. 5, pages 459–473, 1994.
26. S. A. Vanstone, R. C. Mullin, G. B. Agnew. Elliptic curve encryption systems. US Patent 6,141,420, Filed: Jan. 29, 1997.
27. A. Young. Kleptography: Using Cryptography Against Cryptography. PhD Thesis, Columbia University, 2002.
28. A. Young, M. Yung. The Dark Side of Black-Box Cryptography, or: Should we trust Capstone? In *Advances in Cryptology—Crypto '96*, pages 89–103, 1996.
29. A. Young, M. Yung. Kleptography: Using Cryptography Against Cryptography. In *Advances in Cryptology—Eurocrypt '97*, pages 62–74, 1997.

A Security

In this section we prove that indistinguishability and confidentiality holds. Not surprisingly, indistinguishability holds under the ECDDH assumption. The reduction algorithms utilize *point halving* (where we are interested in halving point B to obtain a point C having composite order where $B = 2C$).

A.1 Indistinguishability

The proof below randomizes 3-tuples (see [5]).

Claim 1: (Random Oracle Model) If ECDDH is hard then the SETUP produces prime pairs that are computationally indistinguishable from RSA prime pairs.

Proof: (Sketch) It will be shown that if the SETUP produces primes that are computationally distinguishable from pairs of RSA primes then ECDDH is easy. So, let $\mathcal{D}$ be a distinguisher that distinguishes pairs of primes from the SETUP from pairs of normal RSA primes.

Let $t = \mathsf{GetBlock}(n_s, 0, \theta)$ where n_s is $p_1 q_1$ represented as a binary string. Consider $s = \mathsf{GetBlock}(\pi_\theta^{-1}(t), \theta - (m+1), m+1)$. Note that s is on exactly one of the two twisted curves. There are 3 possibilities. Either,

1. $\mathcal{D}$ distinguishes with non-negligible advantage for such points s on $E_{0,b}(\mathbb{F}_{2^m})$ but with negligible advantage when such points s are on $E_{1,b}(\mathbb{F}_{2^m})$, or
2. $\mathcal{D}$ distinguishes with negligible advantage for such points s on $E_{0,b}(\mathbb{F}_{2^m})$ but with non-negligible advantage when such points s are on $E_{1,b}(\mathbb{F}_{2^m})$, or
3. $\mathcal{D}$ distinguishes with non-negligible advantage for such points s that are on $E_{0,b}(\mathbb{F}_{2^m})$ or $E_{1,b}(\mathbb{F}_{2^m})$.

Without loss of generality suppose that case (1) holds (a similar reduction argument holds for case (2)). For case (1) it follows that $\mathcal{D}$ distinguishes with probability greater than $\frac{1}{2} + \frac{1}{t_1^{\alpha_1}}$ for some fixed $\alpha_1 > 0$ and sufficiently large t_1.

Consider machine $\mathcal{M}_1$ that takes as input an ECCDH problem instance given by $(A_1, A_2, G_0, m, b) = (a_1 G_0, a_2 G_0, G_0, m, b)$ where G_0 has order q_0 and $a_1, a_2 \in \{1, 2, ..., q_0 - 1\}$. The problem is to compute $a_1 a_2 G_0$.

$\mathcal{M}_1(A_1, A_2, G_0, m, b)$:
Input: points A_1, A_2 each having order q_0 that are on curve $E_{0,b}(\mathbb{F}_{2^m})$,
base point G_0, EC parameters m and b
Output: point P having order q_0 on curve $E_{0,b}(\mathbb{F}_{2^m})$
1. choose $u_1, u_2 \in_R \{1, 2, ..., q_0 - 1\}$ and $\mu \in_R \{0, 1, 2, 3\}$
2. set $(B_1, B_2) = (u_1 A_1, u_2 A_2)$
3. set $C_1 = B_1$
4. solve for C_2 in $B_1 = 2C_2$ such that C_2 has order $2q_0$
5. choose C_3 in $C_2 = 2C_3$ randomly (C_3 has order $4q_0$)
6. if $\mu = 1$ then set $C_1 = C_2$
7. if $\mu \in \{2, 3\}$ then set $C_1 = C_3$
8. set $Y_0 = B_2$
9. set $s_{pub} = \mathsf{PointCompress}(E_{0,b}, C_1)$
10. choose s_{priv} to be a random compressed point on $E_{0,b}(\mathbb{F}_{2^m})$ having order q_0
11. randomly choose a base point G_1 having order q_1 that is on $E_{1,b}(\mathbb{F}_{2^m})$
12. choose $x_1 \in_R \{1, 2, ..., q_1 - 1\}$ and set $Y_1 = x_1 G_1$
13. compute $(p_1, q_1) = \mathsf{GetPrimes}_{\mathsf{N},\mathsf{e}}(s_{pub}, s_{priv})$
14. set L to be the empty list
15. step through the operation of $\mathcal{D}(p_1, q_1, G_0, G_1, Y_0, Y_1, m, b)$ while trapping all calls to $\mathcal{R}$, and for each call to $\mathcal{R}$, add the argument to $\mathcal{R}$ to L
16. if L is non-empty then
17. choose $s \in_R L$ and compute $(P, w) = \mathsf{PointDecompress}(E_{0,b}, s)$
18. if $(w = 1)$ then

19. if P has order q_0 then output $(u_1u_2)^{-1}P$ and halt
20. output a random point with order q_0 on $E_{0,b}(\mathbb{F}_{2^m})$ and then halt

Clearly the running time of $\mathcal{M}_1$ is efficient. Note that $\mathcal{D}$ makes at most a polynomial number of calls to random oracle $\mathcal{R}$. So, the number of elements in L is at most $p_2(m)$ where p_2 is polynomial in m.

Consider the algorithm $\mathcal{M}_2$ that takes as input an ECDDH problem instance $(A_1, A_2, A_3, G_0, m, b) = (a_1G_0, a_2G_0, a_3G_0, G_0, m, b)$ where G_0 has order q_0 and $a_1, a_2, a_3 \in \{1, 2, ..., q_0 - 1\}$.

$\mathcal{M}_2(A_1, A_2, A_3, G_0, m, b)$:
1. choose $u_1, u_2, v \in_R \{1, 2, ..., q_0 - 1\}$ and $\mu \in_R \{0, 1, 2, 3\}$
2. set $(B_1, B_2, B_3) =$
$$(vA_1 + u_1G_0, A_2 + u_2G_0, vA_3 + u_1A_2 + vu_2A_1 + u_1u_2G_0)$$
3. set $C_1 = B_1$
4. solve for C_2 in $B_1 = 2C_2$ such that C_2 has order $2q_0$
5. choose C_3 in $C_2 = 2C_3$ randomly (C_3 has order $4q_0$)
6. if $\mu = 1$ then set $C_1 = C_2$
7. if $\mu \in \{2, 3\}$ then set $C_1 = C_3$
8. set $Y_0 = B_2$
9. set $s_{pub} = \mathsf{PointCompress}(E_{0,b}, C_1)$
10. set $s_{priv} = \mathsf{PointCompress}(E_{0,b}, B_3)$
11. randomly choose a base point G_1 having order q_1 on curve $E_{1,b}(\mathbb{F}_{2^m})$
12. choose $x_1 \in_R \{1, 2, ..., q_1 - 1\}$
13. set $Y_1 = x_1G_1$
14. compute $(p_1, q_1) = \mathsf{GetPrimes}_{\mathsf{N},\mathsf{e}}(s_{pub}, s_{priv})$
15. return $\mathcal{D}(p_1, q_1, G_0, G_1, Y_0, Y_1, m, b)$ and halt

Clearly the running time of $\mathcal{M}_2$ is efficient. Observe that if (A_1, A_2, A_3) is an EC Diffie-Hellman triple then (B_1, B_2, B_3) is an EC Diffie-Hellman triple. If (A_1, A_2, A_3) is not an EC Diffie-Hellman triple then (B_1, B_2, B_3) is a random 3-tuple. If the input is not an EC Diffie-Hellman triple then with probability $(1 - \gamma_1(m))$ the tuple (B_1, B_2, B_3) will not be an EC Diffie-Hellman triple. Here γ_1 is a negligible function of m. Thus, with overwhelming probability (B_1, B_2, B_3) matches the input 3-tuple in regards to being a DH triple or not.

Let $1 - \gamma_0(k_0)$ denote the probability that s (recall that we are considering case (1)) corresponds to the EC Diffie-Hellman key exchange value.[7] Here γ_0 is a negligible function of k_0.

Let p_{trap} denote the probability that $\mathcal{D}$ calls $\mathcal{R}$ with the DH shared secret corresponding (B_1, B_2). There are two possible cases. Either, (1) $p_{trap} > \frac{1}{t_2^{\alpha_2}}$ for some fixed $\alpha_2 > 0$ and sufficiently large t_2, or (2) $p_{trap} \leq \gamma(m)$ where γ is a negligible function of m. If it is case (1) then $\mathcal{M}_1$ solves the elliptic curve Diffie-Hellman problem with probability at least $\left(\frac{1}{2} + \frac{1}{t_1^{\alpha_1}}\right) \frac{1}{p_2(m)}(1 - \gamma_0(k_0))\frac{1}{t_2^{\alpha_2}}$. If it is case (2) then $\mathcal{M}_2$ solves the ECDDH problem with probability at least

[7] A borrow bit can be taken and prevent s from being the correct point.

$(1-\gamma(m))\,(1-\gamma_0(k_0))\,(1-\gamma_1(m))\left(\frac{1}{2}+\frac{1}{t_1^{\alpha_1}}\right)$. So, at least one of the machines in the set $\{\mathcal{M}_1,\mathcal{M}_2\}$ can be used to solve ECDDH efficiently. $\diamond$

Claim 1 proves that Property 1 holds.

A.2 Confidentiality

Claim 2: (Random Oracle Model) If the factorization and the EC Computational Diffie-Hellman (ECCDH) problems are hard then confidentiality of the SETUP holds.

Proof: (Sketch) It will be shown that if the confidentiality of the SETUP does not hold then factoring or ECCDH is easy. Let $\mathcal{A}$ be an algorithm that foils the confidentiality of the SETUP with non-negligible probability. More specifically, with probability greater than $\frac{1}{t_1^{\alpha_1}}$ for some fixed $\alpha_1 > 0$ and sufficiently large t_1, $\mathcal{A}(n, G_0, G_1, Y_0, Y_1, m, b)$ returns a prime factor p_1 that divides n.

Consider the following efficient algorithm.

$\mathcal{M}_{1,0}(A_1, A_2, G, m, b)$:
Input: points A_1, A_2 with order q_0 on curve $E_{0,b}(\mathbb{F}_{2^m})$, base point G_0 having order q_0 on curve $E_{0,b}(\mathbb{F}_{2^m})$, EC parameters m and b
Output: A point P on $E_{0,b}(\mathbb{F}_{2^m})$ having order q_0
1. choose $u_1, u_2 \in_R \{1, 2, ..., q_0 - 1\}$ and $\mu \in_R \{0, 1, 2, 3\}$
2. set $(B_1, B_2) = (u_1 A_1, u_2 A_2)$
3. set $C_1 = B_1$
4. solve for C_2 in $B_1 = 2C_2$ such that C_2 has order $2q_0$
5. choose C_3 in $C_2 = 2C_3$ randomly (C_3 has order $4q_0$)
6. if $\mu = 1$ then set $C_1 = C_2$
7. if $\mu \in \{2, 3\}$ then set $C_1 = C_3$
8. set $Y_0 = B_2$
9. set $s_{pub} = \mathsf{PointCompress}(E_{0,b}, C_1)$
10. choose s_{priv} to be a random compressed point on $E_{0,b}(\mathbb{F}_{2^m})$ having order q_0
11. randomly choose a base point G_1 having order q_1 on curve $E_{1,b}(\mathbb{F}_{2^m})$
12. choose $x_1 \in_R \{1, 2, ..., q_1 - 1\}$
13. set $Y_1 = x_1 G_1$
14. compute $(p_1, q_1) = \mathsf{GetPrimes}_{\mathsf{N,e}}(s_{pub}, s_{priv})$
15. set $n = p_1 q_1$ and let L be the empty list
16. step through the operation of $\mathcal{A}(n, G_0, G_1, Y_0, Y_1, m, b)$ while trapping all calls to $\mathcal{R}$, and for each call to $\mathcal{R}$, add the argument to $\mathcal{R}$ to L
17. if L is non-empty then
18. choose $s \in_R L$ and compute $(P, w) = \mathsf{PointDecompress}(E_{0,b}, s)$
19. if $(w = 1)$ then
20. if P has order q_0 then output $(u_1 u_2)^{-1} P$ and halt
21. output a random point on $E_{0,b}(\mathbb{F}_{2^m})$ having order q_0 and then halt

The size of list L is at most $p_2(m)$ where p_2 is polynomial in m. A similar reduction algorithm $\mathcal{M}_{1,1}$ can be constructed for the elliptic curve in which

the value $a = 1$. The remainder of this proof considers EC Diffie-Hellman over $E_{0,b}(\mathbb{F}_{2^m})$ unless otherwise specified. Now consider the following algorithm.

$\mathcal{M}_2(n)$:
1. randomly choose a base point G_0 having order q_0 on curve $E_{0,b}(\mathbb{F}_{2^m})$
2. randomly choose a base point G_1 having order q_1 on curve $E_{1,b}(\mathbb{F}_{2^m})$
3. choose $x_0 \in_R \{1, 2, ...q_0 - 1\}$ and choose $x_1 \in_R \{1, 2, ..., q_1 - 1\}$
4. compute $Y_0 = x_0 G_0$ and $Y_1 = x_1 G_1$
5. output $\mathcal{A}(n, G_0, G_1, Y_0, Y_1, m, b)$

Clearly the running time of $\mathcal{M}_2$ is efficient.

Let $t = \mathsf{GetBlock}(n_s, 0, \theta)$ where n_s is $p_1 q_1$ represented as a binary string. Consider $s = \mathsf{GetBlock}(\pi_\theta^{-1}(t), \theta - (m+1), m+1)$. Let $1 - \gamma_0(k_0)$ denote the probability that s corresponds to the EC Diffie-Hellman key exchange value. Here γ_0 is a negligible function of k_0.

Let p_{trap} denote the probability that $\mathcal{A}$ calls $\mathcal{R}$ with the DH shared secret corresponding to (B_1, B_2). One of the following must occur: (1) $p_{trap} > \frac{1}{t_2^{\alpha_2}}$ for some fixed $\alpha_2 > 0$ and sufficiently large t_2, or (2) $p_{trap} \leq \gamma(m)$ where γ is a negligible function of m. If it is case (1) then $\mathcal{M}_{1,0}$ (or $\mathcal{M}_{1,1}$) solves ECCDH with probability at least $\frac{1}{t_1^{\alpha_1}} \frac{1}{t_2^{\alpha_2}} (1 - \gamma_0(k_0)) \frac{1}{p_2(m)}$. If it is case (2) then $\mathcal{M}_2$ factors with probability at least $(1 - \gamma(m)) \frac{1}{t_1^{\alpha_1}}$ that is equal to $\frac{1}{t_1^{\alpha_1}} - \frac{\gamma(m)}{t_1^{\alpha_1}}$. It follows that ECCDH or factoring is efficiently solvable. ◇

Claim 2 proves that Property 2 of a strong SETUP holds. So, we have:

Theorem 1. $(\mathsf{KleptoKeyGen_{N,e}}, \mathsf{KleptoRecoverKey_{N,e}})$ *is a* **strong SETUP**.