A Survey of ESIGN: State of the Art and Proof of Security

Richard A. Kramer
Oregon State University
kramerri@onid.oregonstate.edu

UPDATED: WINTER TERM 2017

Abstract

RSA [1] generates digital signatures and cipher text, *S*, by performing exponentiation on a message, *M*, to the e-th power of the form M^e mod(*n*), where *n* is based on large prime numbers. RSA has been proven to be fundamentally secure, given the complexity of the e-th root mod(*n*) problem that RSA presents. Unfortunately, RSA is computationally intensive, especially with regards to generating keys/digital signatures and decrypting cipher text messages. These factors are a major drawback for RSA, especially when RSA is implemented within embedded solutions that have limited processing capabilities such as smart cards. To address this problem, ESIGN [2] was developed as an improvement to RSA, whereas ESIGN offers improved computational complexity, yet maintains the complexity of the e-th root mod (*n*) problem that provides security [9].

This survey provides an in-depth review of: (Section 1) basic ESIGN and how it works, (Section 2) attempts to improve ESIGN (and its security) via the introduction of ESIGN variants and other recommendations, (Section 3) the continually evolving quest to prove and improve ESIGN's security. This survey then concludes with Section 4.

Keywords

RSA, ESIGN, cryptography, digital signature, encryption, decryption, factoring

TABLE OF CONTENTS

Section Page

LIST OF FIGURES

Figure Page

LIST OF TABLES

Table Page

Section 1: Introduction

RSA (named after RSA's inventors, Rivest, Shamir, and Adelman) was introduced in 1977 and was further published in 1978 [1]. RSA generates digital signatures and cipher text, S, by performing exponentiation on a message, M, to the e-th power of the form M^e mod(n), where n is based on large prime numbers. Through the years, RSA has been proven to be fundamentally secure, given sufficient complexity for the e-th root mod(n) problem. Because of the way RSA implemented this scheme, the generation of RSA keys/digital signatures and RSA decryption are computationally complex.

The foundation of RSA is asymmetrical cryptography were n (part of the Public Key (PK)) = pq (which forms the Secret Key (SK)), where p and q are (preferably large) prime numbers. The cryptography scheme for RSA is further explained in Section 1.1. Unfortunately the generation of RSA keys/digital signatures and decryption of RSA messages is often-times performed in remote devices such as smart phones or smart cards, where processing resources are most scarce. This dilemma gave rise to a need for a less computationally demanding e-th root mod (n) based security solution. ESIGN's creation filled this void in 1985. ESIGN builds on RSA and added some additional schemes to significantly reduce the computational complexity of generating ESIGN keys/digital signatures and the verification of signatures, employing $n = p^r q$ among other changes[1], whereas r is typically, r=2 [2]. The cryptography scheme of ESIGN and an explanation on why ESIGN is less computationally intensive is explained in Sections 1.1 and 1.2.

Based on the creation of ESIGN and security threats, a number of improvements have been made to ESIGN since ESIGN's inception. These improvements are discussed in Section 2 [3, 6, 7, 8, 13].

To prove ESIGN's security, methods to challenge ESIGN's security have continued to evolve [2, 3, 4, 5, 7, 9, 10, 11]. With the evolving attempts to prove ESIGN's security, numerous recommendations have been made to increase ESIGN's security by making the e-th root mod(n) problem increasingly complex to offset the ever increasing computing power and attacks of potential adversaries. The methods used to continually prove ESIGN's security are presented in Section 3.

Finally, Section 4 provides the conclusion including potential future applications [12]. For the aid of the reader, a glossary of terms is provided as an Appendix.

[1] For example, beyond the changes in how computations are performed, ESIGN shifts some of the computational processing such that it can be done independent of the actual messages being digitally signed.

1.1 The RSA Cryptography Scheme [1].

RSA encryption and decryption follows the form of:

Choose and calculate: 1. Choose p and q = prime numbers 2. From that, calculate $n = pq$ and $\phi(n) = (p-1)(q-1)$ 3. From that, choose e (relatively prime) to $\phi(n)$
Public Key (PK = *n, e*): n, from above e, from above
Secret Key (SK = *p, q, d*): p and q from above $d = e^{-1} (\mathrm{mod}(\phi(n))$. In other words, $e\ d \equiv 1\ \mathrm{mod}(\phi(n))$

Figure 1: RSA Encryption Scheme

From the above, to encrypted cipher text C, using message M, the following operation is performed:

$$C = M^e \bmod(n)$$

To decrypt the cipher text C to recover message M, the following operation is performed:

$$M = C^d \bmod(n)$$

Alternatively, the secret key SK element d can be used to encrypt M, and in return, public key PK element e can be used to recover M from the cipher text C.

The computational complexity of RSA is in part, a result of the requirement to calculate the Secret Key (SK) element d, which requires the inverse exponentiation of e of the form $d = e^{-1} (\mathrm{mod}(\phi(n))$, which relates to both the calculation of the Secret Key (SK) element d and decryption of the cipher text message via $M = C^d \bmod(n)$.

1.2 The Basic ESIGN Cryptography Scheme [2].

ESIGN builds upon RSA by introducing the relationship $n = p^2q$ and other schemes as follows:

Choose and calculate[2]: 1. Choose *p and q* = large prime numbers where $p > q$ 2. From that, calculate $n = p^2q$ which results in a $3k$ bits long value (note: in [2], "*n*" is denoted a "*N*") 3. Choose *e*, a positive integer ≥ 4 (note: in [2] "*e*" is denoted as "*K"*) 4. Choose k: The use of *k,* was later formalized to be the bit length of *p* and/or *q* [3, 9, 10 as examples]
Public Key (PK = *n, e, k*): *n*, from above *e*, from above *k,* was later added, as discussed above While not a specific part of the Public Key (PK) the hash function H must be agreed upon in advance.
Secret Key (SK = *p, q*): *p* and *q,* chosen above

Figure 2: ESIGN Basic Encryption Scheme

ESIGN SIGNATURE:

From the above, *digital signature generation* is performed as follows:

1. Choose *r*: Pick a random number $r \in Z^*_n$, where Z^*_n denotes a set of integer numbers between 0 and *n*-1 which is relatively prime to *n* (*n*ote: in [3], "*r"* is referred to as "*X*")

2. Calculate variables that will be used to later create *S* (the digital signature) as follows:
   a. $Z = H'(M)) - r^e \bmod(n)$ where H is a one-way hash function $H(M) \in Z^*_{pq}$ for any positive integer *M* and H'(*M*) is a Trisection Hash of H(*M*) (see Section 2.2 – below)
   b. $\omega_0 = \lceil [(H'(M)) - r^e \bmod(n)] / pq \rceil$, (also called "*W" in* [2] which is rounded to the upper integer ($\lceil\ \rceil$))

[2] Based on [3]. As will be discussed further below, after the initial introduction of ESIGN, further restrictions on the value of k (bit length for *p* and *q,* the minimum value of *e* where later added.

c. It was later formalized [3, 9, 10] that:

i. $\omega_1 = \omega_0 pq - Z$

ii. If $\omega_1 > 2^{2k-1}$, then repeat the above steps starting with picking a new value *r*.

d. $Y = [\omega_0/[(er^{e-1})] \bmod(p)$

3. $S = r + Ypq$ (the digital signature)

An important attribute of ESIGN, over RSA, is the fact that many of the computationally intensive steps for signature can be calculated, independent of the message *M,* such as the following operations from above, shown in **BOLD/underline**:

$\omega_0 = \lceil ((H(M)) - \mathbf{\underline{r^e \bmod(n))/pq}} \rceil$, also call "*W" in* [1]

$Y = [W/\mathbf{\underline{(er^{e-1})]\bmod(p)}}$

ESIGN SIGNATURE VERIFICATION:

For *digital signature verification*, uniquely, *ESIGN provides verification based on a range* (versus an exact value as in RSA), by solving the following inequality:

1. $0 \leq S < n$

2. $0 \leq S^e \bmod(n) / 2^{2k} < s^{(k-1)}$

3. H'(M) computed = H'(M) received.

Further, for embedded applications, such as smart cards where *p* and *q* were fixed for the life of the card, pre calculations could be performed that use *p* and *q* (e.g., *n*), and the answers stored in a look-up table in EEPROM at the time of production, versus calculating these values via a microprocessor on the fly [2].

The above combination of attributes related to ESIGN makes ESIGN computationally advantageous over RSA. At the time, it was estimated that ESIGN was more than "twenty times faster than that of RSA" including the "signature generation of about 0.2 seconds" [2, at Abstract].

As shown above, when contrasting RSA to ESIGN, in addition to avoiding inverse exponentiation, ESIGN also advantageously allows for the pre-processing of certain variables that are independent to the message, *M*, and ESIGN verifies the digital signature based on a *range* of values rather than a specific value.

Section 2: ESIGN Implementations, Improvements and Variants

2.1 ESIGN Implementations, Improvements and Variants Overview.

As discussed above in Section 1.2, ESIGN was created in 1985 [2]. Shortly after ESIGN's introduction, ESIGN was found to be vulnerable to attack for $e \leq 4$ [2 at pg. 449, Section 2.3]. Since that time, other improvements have been recommended for ESIGN. These improvements include the creation of the following variants/extensions of ESIGN:

1. **TSH-ESIGN –** which was adopted as part of the ISO/IEC 14888-3 [3] standard
2. **ESIGN-D -** ESIGN-Deterministic [6]
3. **ESIGN-R-** ESIGN-Randomized [6]
4. **Other Variants and Recommendations for ESIGN** [7, 8]
5. **ESIGN-PSS** - ESIGN-Probabilistic Signature Scheme [13]
6. **ESIGN-PSS-R**- ESIGN-Probabilistic Signature Scheme with Recovery [13]

Each of these ESIGN variants/extensions are further discussed below in the following subsections.

2.2 TSH-ESIGN [3]

TSH-ESIGN (Trisection Size Hash) [3] was adopted as an appendix of ISO/IEC 14888-3. Overall, in TSH-ESIGN p and q at set to value k bits in length, thus the length of $n = p^2q$ is $3k$ bit long. It is from this relationship of "*3k*" that the trisection hash is formulated (*e.g.,* 3 sections). For TSH-ESIGN, the hash function H is such that the H'(M) must be of the proper length, thus resulting in: H'(M) = (0||H(M)), where the hash input for Z discussed above is (H'(M)||0^{2k}), thus Z = (0||H(M)||0^{2k}).

The security of TSH-ESIGN is proven given a sufficiently hard n factoring problem. Thus, TSH-ESIGN is EUA-CMA (Existentially Unforgeable Against a [adaptive] Chosen Message Attack) under the AERP (Approximate e-th Root Problem) assumption.

> *The **Approximate e-th Root Problem (AERP) assumption** is: there is no efficient algorithm for solving the AERP, whereas the AERP problem entails finding the e-th root of a modulo n equation.*

Further, TSH-ESIGN retains the computational advantage over RSA. For the comparison below, RSA and ESIGN are contrasted where $e = 2^{16+1}$ for RSA, and $e = 2^5$ for TSH-ESIGN. The significant reduction of computations for TSH-ESIGN is shown below:

Table 1: Comparison of computational amount [3, Table 2]

Schemes	*Sig. Gen. (M(1024))*	*Sig. Ver. (M(1024))*
TSH-ESIGN	9	5
RSA-based scheme (e.g., PSS or FDH-RSA)	384	17
EC-based scheme (e.g., EC-Schnorr or EC-DSA)	24	28

2.3 ESIGN-D (Deterministic) [6]

While ESIGN is proven to be e-th root mod(n) secure (*e.g.,* under the AERP assumption), a number of security proofs for ESIGN have been flawed (*e.g., see* [4] pg. 3, Section 1.4). To extend the provability of ESIGN, a *deterministic* variant of ESIGN was created called ESIGN-D (Deterministic) [6].

To create the ESIGN-D variant, the following steps (or modified steps) are performed relative to the original ESIGN implementation shown in Section 1.2 above:

Integral to the ESIGN-D variant is:

1. Instead of choosing the random number r, a one-way function ϕ is created. The function ϕ outputs a randomly distributed number that is mod(n).
2. A new k bit value Δ is created and is included as part of the Secret Key (SK).
3. Instead of using the iterative calculation of ω_1 (where from above, $\omega_1 = \omega_{0pq} - Z$, where ω_0 is dependent on r; $\omega_0 = \lceil ((H(M)) - r^e \bmod(n))/pq \rceil$ and if $\omega_1 > 2^{2k-1}$ then a new r is chosen), instead of r, the following new variable is introduced, which I will call r'.

$$r' = \phi(H(M)||\Delta||i) \text{ for } i = 0,1,2,\ldots$$

Simulation results show that $S^e \bmod(n) \bmod(2^{2k-1})$ is indistinguishable from the value generated from the signature algorithm, where Δ is sufficiently large such that ϕ is unpredictable.

Overall, ESIGN-D adds additional processing costs, yet at bit lengths of 1152 or 1536 there are no additional security gains as opposed to unmodified ESIGN (*see* [6] at pg. 6, Section 2.4).

2.4 ESIGN-R (Randomized) [6]

ESIGN-R (ESIGN-Randomized) was created to allow for the simulation of the probabilistic output of the ESIGN signature generator (oracle). Accordingly, the new signature algorithms for ESIGN-R are as follows:

1. Generate additional random number ρ, where the length of ρ is $\leq 2\log_2(qS)$
2. $H'(M) = H(M||\rho)$ where the hash output H' is the result of the combination of the hash H of the message M and ρ
3. $\omega_0 = \lceil ((H'(M))2^{2k} - r^e \bmod(n))/pq \rceil$, where as compared to standard ESIGN as described in Section1.2, the hash value $H(M)$ is replaced with $H'(M)$ and multiplied by 2^{2k}
4. The enhanced digital signature is calculated as $\sigma = M||\rho||S$

The digital signature verification is valid if:

$H'(M) = \lfloor S^e/2^{2k} \rfloor$, where $H'(M) = H(M||\rho)$

Overall, ESIGN-R adds both additional processing costs and bit length due to the addition of the variable ρ, yet at bit lengths of 1152 or 1536 there is no additional security gains as opposed to unmodified ESIGN (*see* [6] at pg. 6, Section 2.4).

2.5 Other Variants and Recommendations for ESIGN [7, 8]

A number of other variants have been provided to improved ESIGN, including variants to RSA that can be extended to ESIGN. A number of these variants include:

1. Recommendations for the size of e and k [3,4]

   As discussed above [3], a minimum size of $e \geq 4$ was recommended based on the fact that $e = 2, 3$ was found to be venerable [3]. Upon evaluation of the e-th root problem , it was recommended that $k \geq 320$ bits (*see* [4] at pgs. 4-5) and Section 3 below).

2. Batch RSA Extended to ESIGN [8]

   The concept of batch processing of, for example multiple decryptions (or digital signatures, or signature verifications) was applied to RSA, specifically because RSA requires modulo exponential inversion to recover the original message (*e.g.,* $M = C^d \bmod(n)$ where $ed \equiv 1 \bmod(n)$) [8]. The requirement for modulo exponential inversion is mitigated in ESIGN, which reduces computational requirements. Nonetheless, based on implementation, opportunities exist to batch process certain portions of the ESIGN algorithms for multiple messages, such as the batch selection of large prime numbers p and q and random number selection r sets, to be cued for future multiple message processing.

3. Multi Key Settings [4]

   As an extension of ESIGN to prevent an attacker from forging keys at each level within a multi key setting, the hash value H(M) can be modified to be H'(M) such that *the message M is hashed with a public key at each level* of security within a multi key environment [4], thus:

   $H'(M) = H(M||PK_x)$ where x is the Public Key at a specific level, x

4. Proposed Security Levels for ESIGN [7]

   To continue to increase the difficulty of the AERP, two levels of security have been recommended for the length of k, including [7] at pg. 7, Section 6 as follows:

   $k_{Level\_1} = 384$ bits
   $k_{Level\_2} = 768$ bits
   Where $e = 1024$

2.6 ESIGN-PSS (Probabilistic Signature Scheme) [13].

As ESIGN continues to evolve and benefit from the advancements of RSA, ESIGN-PSS (ESIGN-Probabilistic Signature Scheme) was added to the ISO/IEC14888-2:2008 standard.

With regards to key generation, the primary difference between the baseline ESIGN scheme described above in Section 1.2, and ESIGN-PSS scheme is the addition of the variable v which is chosen as follows:

$8 \leq v$, 2^{3k-1}

GCD(v, n) =1

GCD(v, $\phi(pq) \leq 3k$

From this, the following is the Public Key (PK) and the Secret Key (SK) for ESIGN-PSS:

Public Key (PK = *n, k, v*): Where *n, k* are comparable to the values in the previously discussed for ESIGN (*see* Section 1.2).
Secret Key (SK = *p, q, k, v*): Where *p, q*, and *k* are comparable to the values in the previously discussed for ESIGN (*see* Section 1.2). $n = pq$, where in [13] these variables are called *p1* and *p2* respectively and *n* is said to be part of the Secret Key (SK).

Figure 3: ESIGN-PSS Keys [13]

The biggest changes from ESIGN to ESIGN-PSS relates to the digital signature generation (and verification). Specifically, ESIGN-PSS adds a random salt E, where E is then combined with the hash of the message M to form the digital signature. In fact, for ESIGN-PSS, the hashed message M is hashed (again) using hash function *h:*

$$HH = h(0^{64}||H(M)||E)$$

and then hashed again using hash function g :

$$R = g(HH) \oplus (0^{kg-kE}-1||1||E).$$

For the digital signature verification, this process is reversed using the Public Key (PK). For a full description of the algorithms used, see reference [13] at pg. 1396, Sections "Signing Algorithm, **Sig**" and "Verification Algorithm, **Ver**".

2.7 ESIGN-PSS-R (Probabilistic Signature Scheme with Recovery) [13].

ESIGN-PSS-R (ESIGN-Probabilistic Signature Scheme with Recovery) builds on the abovementioned ESIGN-PSS [13] and further provides a signature scheme *with message*

recovery. The Public Key (PK) and Secret Key (SK) are identical to those used in the ENSIGN-PSS scheme.

Additionally, ESIGN-PSS-R provides a *recoverable message length*, defined as follows:

Len_{M1}, where $Len_{M1} < k^g - k^E - 1$

Further, the ESIGN-PSS digital signature generation algorithm is modified going from ESIGN-PSS to ESIGN-PSS-R to include the additional parameter message recovery parameter Len_{M1} within the respective hash functions used in the ESIGN-PSS scheme:

$$HH = h(Len_{M1}||M_1||H(M_1)||E)$$
$$R = g(HH) \oplus (0^{kg-kE}-1||1||E)$$

For the digital signature verification, this process is reversed using the Public Key (PK). For a full description of the algorithms used, see reference [13] at pg. 1403, Section "Signing Algorithm, **Sig**" and at pg. 1404, Section "Verification Algorithm, **Ver**".

Section 3: ESIGN's Proof of Security

Overall, the objective to a digital signature scheme is to be Existentially Unforgeable Against a Chosen Message Attack (EUA-CMA). The goal of an adversary, *A*, is to obtain a valid signature on a single message *M*, after having obtained a collection of prior messages before *M*, of *A*'s choice [4]. As discussed in Section 3.1 immediately below, the core of ESIGN's approach to EUA-CMA is that the AERP is sufficiently intractable [4].

3.1 ESIGN Proof of Security Overview and the AERP (Approximate e-th Root Problem).

The core of ESIGN's security is related to the ability to <u>*not solve*</u> the Approximate e-th Root Problem (AERP) and the AERP assumption as discussed above.

Numerous schemes have been developed to attempt to efficiently solve the AERP problem. Further, these schemes have been developed under varying assumptions related to the core complexity of the AERP including but not limited to: (1) the complexity of the e-th root problem based on the value of *e*; (2) the size of *p, q, n*; (3) potential leakages of information; (4) non-randomness / bias, and various other attack scenarios. This section discusses the various methods and schemes used to challenge and prove ESIGN's security.

In general, a multitude of schemes and assumptions related to factoring (and the AERP assumption) have been used to challenge the security of ESIGN. These scheme and methods include:

1. **Lattice Based Reduction Factorization and LLL (Lenstra, Lenstra, Lovasz)** [5, 7, 9, 10, 11, 14]
2. **Quadratic Sieve Factoring** [4]
3. **ECF / ECM (Elliptic Curve Factoring / Elliptic Curve Method)** [4,5]
4. **NFS (Number Field Sieve)** [4,5]

Each of these methods and their effectiveness is further discussed below:

3.2 Lattice Based Reduction Factorization and LLL (Lenstra, Lenstra, Lovasz) [5, 7, 9, 10, 11, 14]

Simply stated, lattice based factorization is a method of building a matrix made up of portions of the polynomial to be factored, and then using the output of a factorization table to complete the matrix and thus find the factors. As the complexity of the polynomial equation increases (*e.g.*, the value of *e*), the complexity of the lattice increases to the point of unmanageability. The LLL (Lenstra, Lenstra, Lovász) algorithm significantly improved lattice based factorization by reducing the lattice using short vectors in polynomial time (*see* [10]; *also see* discussion for [14] below).

Related to ESIGN, lattice based factorization was proven effective against factoring early ESIGN implementations where the exponent of the random value r was small (*e.g.,* $e = 2$ or 3) [2]. For lattice based reduction factorization, with small values of *e,* lattice based structures are manageable and the method is efficient [7], yet at large values of *e* (*e.g.*, $e \geq 4$ or ≥ 8), factorization using lattices becomes complex and even impossible [11]. One way to reduce the complexity (thus posing a threat to security) is to have leakage and/or non-randomness bias. Because of the exposed vulnerability of ESIGN using lattice based factorization at $e = 2$ or 3, e was increased to $e \geq 4$, and even later $e \geq 8$ [2, 3, 4, 5, 7, 9, 10, 11].

Further and importantly, using the LLL algorithm: even a small amount of leakage or non-random bias has been proven to be detrimental to determining the ESIGN Secret Key (SK). For instance, the requirement for $r < pq$ has resulted in some ESIGN implementations simply masking the upper bits of r in a manner to force $r < pq$. Such masking introduces bias (*e.g.,* non-randomness) that lattice based factoring is then able to exploit [7]. The use of LLL to exploit other vulnerabilities within ESIGN are further

discussed in Section 3.6 below, entitled "Summary and Vulnerabilities of ESIGN Security".

A discussion of LLL and polynomial time

In the paper "Small secret exponent attach on RSA variant with modulus $N = p^{[j]}q$" [14] (*see* note 1 below), the paper provides an analysis of factoring for the RSA Secret Key (SK) element d for the exponent value $j = 2$. As discussed above, the digital signature for RSA is of the form: $C = M^e \bmod(n)$, and the decryption of the cipher text, C, is the inverse exponentiation of e for verifying the digital signature using the Secret Key (SK) element, d (*e.g.,* $M = C^d \bmod(n)$).

In contrast, for ESIGN $n = p^2q$, and the signature is of the different general form: $S = r + Ypq$ and the verification of the digital signature entails solving the range $H(M) \leq S^e \bmod(n) < H(M) + 2^{2|n|2/3}$ using e. From this, it can be seen that part of the ESIGN polynomial to be factored to verify the digital signature entails r raised to the e-th power (*via* S^e). Thus the factoring RSA's $C^d \bmod(n)$ is not equivalent to factoring ESIGN's $S^e \bmod(n)$ where ESIGN's digital signature includes the use of the random number r raise to an exponent e (not its inverse, where there is no equivalent d as is the case in RSA).

Thus the analysis and proof of the paper [14] that proves factoring for RSA in polynomial time for the decryption exponent, $d \leq N^{0.395}$ cannot be directly applied to ESIGN. With that said, what is true is: *Both RSA and ESIGN are proven to be factorable in polynomial time using LLL* (*see* [14] for RSA, and "The Insecurity of ESIGN in Practical Implementations" [10] for ESIGN).

Thus it is true that factorization in polynomial time applies to both RSA and ESIGN [10, 14].

Note 1: To avoid confusion with the use of random number "*r"* used in ESIGN, I have changed the exponent reference to "*j*".

For LLL for ESIGN, the run time maximum is said to be:

$$\text{Run time (max)}^3 = O(t^4(\log(i \leq t)$$

Where t is the number of linear independent vectors generated in the LLL scheme (*see* [10] at pgs. 497-498).

[3] Here I have changed "*d*" used in [10] to "*t*" to mitigate any confusion with the Secret Key (SK) element *d* used in RSA.

3.3 Quadratic Sieve Factoring [4, 15]

In certain cases, the quadratic sieve factoring method has been proven to be more efficient than the lattice base reduction method, for example, when $e \geq 4$ given the special case of n being the product of two primes of equal size [4]. Nonetheless, other methods have proved more efficient/effective including the Number Field Sieve (NFS) method, thus quadratic sieve factoring has been surpassed by NFS [4, 15].

3.4 ECF / ECM (Elliptic Curve Factoring / Elliptic Curve Method) [4, 5, 15]

The elliptical curve factoring method (ECM) is the fastest factoring algorithm known when the algorithm run time is measured in terms of small prime numbers. This is specifically useful based on the smallest prime of n (*e.g,* q, where in ESIGN by definition $p > q$ as discussed in Section 1.2) assuming that the smallest prime is a small number overall.

By definition, ESIGN requires large prime numbers [2] and in fact later ESIGN implementations stipulated the minimum size of the prime numbers to be of large bit length k, thus making ECF ineffective and inefficient when dealing with large prime numbers for both p and q (*see* [4] at pg. 6, "Summary" and [15] at pg. 211). With the smallest prime number of n being large [4] such that the a minimum bit length of k = 320 bits for p and q (or |n| is at least 1024 bits long, the ECF method is ineffective based on long run times ([4] pg. 5, [5] pg. 3).

3.5 NFS (Number Field Sieve) [4, 5, 15]

NFS(Number Field Sieve) has been validated to be superior based on the work of numerous researchers and experimentation. In fact, it is asymptotically superior to the above mentioned quadratic sieve factoring method. Overall, NFS's expected run time is given as:

$$\text{Run time} = O(exp((1.306 + o(1)) \ln(n)^{1/3}(\ln(\ln(n)))^{2/3})) \text{ [4]}$$

As can be seen from the above, the run time is dependent of n.

3.6 Summary and Vulnerabilities of ESIGN Security

In summary, using a wide variety of methods and schemes, ESIGN and the later enhanced variants of ESIGN have been proven to provide EUA-CMA based on a sufficiently intractable AERP presented to any would be adversary A provided leakage and non-randomness / bias is avoided. In sharp contrast, it has been proven that when

the true randomness / bias of the ESIGN process is compromised, ESIGN becomes insecure [7, 10].

The impact of ESIGN security based on leakage

Using the LLL algorithm with a 1152 bit modulus (n) and thus the random variable r bit length is 768 bits:

> ***IF ONLY 8 BITS of the 768 bits for r are exposed, based on simply obtaining 57 signatures: the Secret Key (SK) can be recovered in a matter of minutes (using 2003 computing speed estimates).*** This can be done by solving $r^e \bmod(n)$ which allows the recovery of p and q [10].

Experimental results of ESIGN vulnerabilities should l bits of leakage occur is as follows:

Table 2: Experimental Results of ESIGN given Leakage of *l* Bits of *r* [10]

$n = 3k$	$2k$	$\log(r)$	experimental value for ℓ	theoretical bound for ℓ	d	time to factor
512	340	335	5	8	55	2 min 10
768	512	506	6	9	55	2 min 20
1024	682	674	8	11	56	2 min 30
1152	768	760	8	11	57	3 min
1536	1024	1013	11	14	57	4 min 10
2048	1364	1349	15	17	57	5 min 50

Where:

l = bits of leakage from random number r.

d = the number of signatures that would need to be received by an adversary A prior to being able to factor p and q.

Section 4: Test Results and Analysis

4.1 Setup and Test Overview

ESIGN as compare to ECDSA (Elliptical Curve Digital Signature Algorithm) and ED25519 (Edwards Curve 25519) were fully implemented and evaluated to compare processing speed for:

- Sign
- Verify
- Sign + Verify

Testing was conducted for N=1, 100, 10,000 and 100,000 messages of 16 bytes in length. Implementation was done in 'c' programming, using gcc.

A summary of the setup conditions for each of signature schemes is as follows:

ESIGN was implemented in 'c' using the M.I.R.A.C.L. libraries according to ESIGN-PSS [13]. The key bit length for n = 3,084 bits, based on each of p and q = 1,024 bits, thus yielding a 3,084 bit signature. The native H(M) hash length was 256 bits and then made into a trisection hash.

ECDSA was likewise implemented in 'c' code using the M.I.R.A.C.L. libraries and using a modified version of M.I.R.A.C.L. reference software for implementing ECDSA. The key and signature size bit lengths was all 192 bits each.

ED25519 was implemented using modified 'c' code using an MIT implementation for x64 bit processors. The key length was 256 bits and the signature length was 2,048 bits. Testing for ED25519 was only done on the below mentioned 64 bit Intel i5 processor.

The testing was conducted on the following platforms:

1) A 64 bit Intel i5-3317U running at 1.70 GHz
2) A Raspberry Pi-3 with a Broadcom BCM2837 Cortex A53 Processor

4.2 Results and Summary

The data taken from the above test setup and platforms indicates superior performance of the ESIGN verification over ECDSA and EC25519 making.

Shown below is the following:

1) Figure 4: Average Processing Time/Message versus Crypto-Scheme (100,000 Messages - 64 bit Intel i5)
2) Figure 5: Average Processing Time/Message versus Crypto-Process (1 to 100,000 Messages - 64 bit Intel i5)
3) Figure 6: Ave. Processing Time/Message versus Processor (100,000 Messages)
4) Figure 7: Average Processing Time/Message versus Crypto-Scheme (100,000 Messages - Broadcom BCM2837 Cortex A53)

From the results, ESIGN outperformed ECDSA overall. For example, on the Raspberry Pi Cortex A53 processor, ESIGN verify versus ECDSA verify outperformed ECDSA by 901% while at the expense of only -12% for ESIGN sign versus ECDSA verify (*see* Fig. 7 – below).

Turning to ESIGN as compared to ED25519, on the Intel i5 64 bit processor, ESIGN verify outperformed ED25519 verify by 930% while at the expense of -77% comparing ESIGN sign to ED25519 sign.

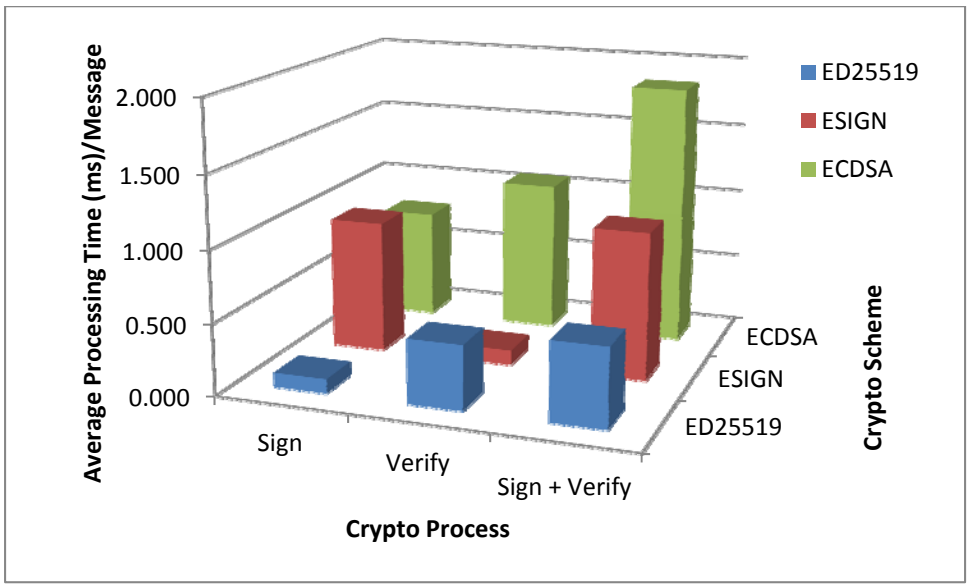

Figure 4: Average Processing Time/Message versus Crypto-Scheme (100,000 Messages - 64 bit Intel i5)

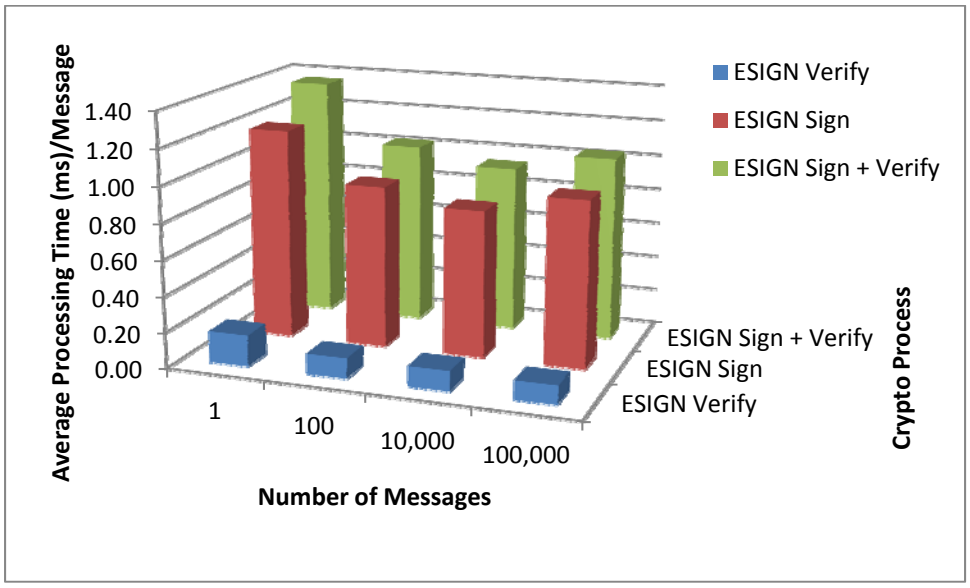

Figure 5: Average Processing Time/Message versus Crypto-Process (1 to 100,000 Messages - 64 bit Intel i5)

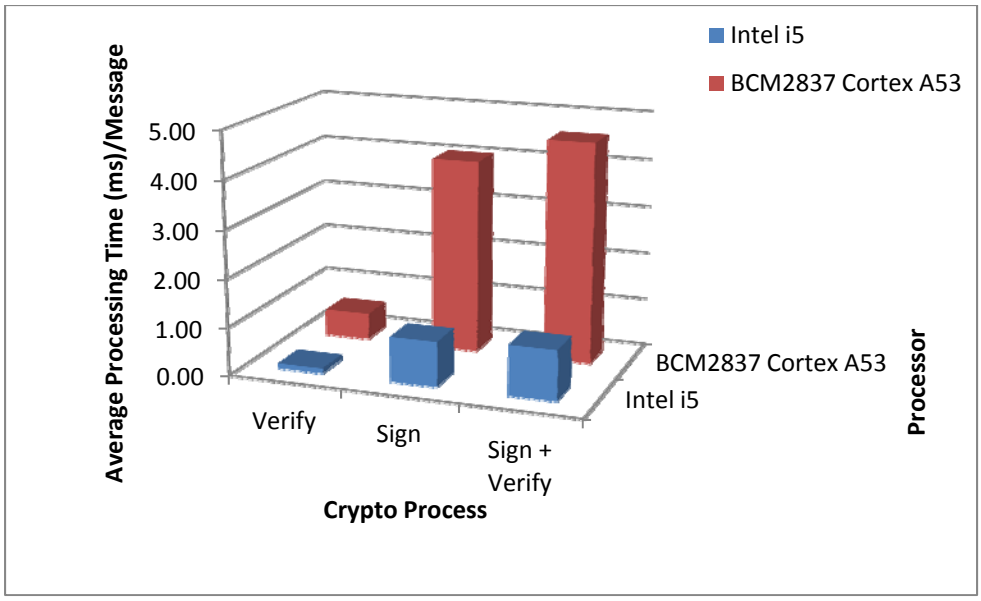

Figure 6: Ave. Processing Time/Message versus Processor (100,000 Messages)

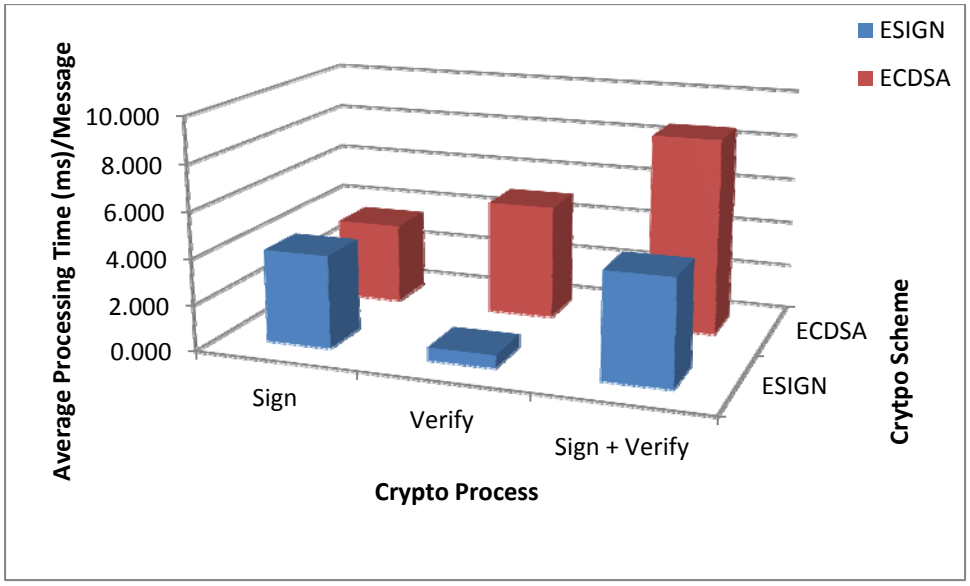

Figure 7: Average Processing Time/Message versus Crypto-Scheme (100,000 Messages - Broadcom BCM2837 Cortex A53)

Section 5: Conclusion

While it is said that the mathematics of the AERP are not fully understood for large prime numbers (at least when e is not prime to $\phi(n)$), numerous proofs have been presented proving the security of ESIGN using a plethora of methods (including those listed above). Consistently ESIGN has been proven secure based on AERP absent of leakage or non-randomness / bias. In fact, ESIGN has been proven to provide near uniform distribution over a large interval [9]. For example, Lemma 6 of reference [9] (see, pgs. 295-300) proves that:

> For a fixed message M, the e-th power S^e mod(n) of the output S is uniformly distributed over the set of e-th powers of elements Z_n^*, lying in interval*:* y, $y + 2^{2k-1}$, (where y = the $0||H(M)||0^{2k}$; *see* discussion for ESIGN-TSH above).

As shown above in Section 3.6, the security of ESIGN is highly compromised in the presence of leakage and/or non-randomness / bias.

Given the favorable computational advantages of ESIGN, future applications of ESIGN are envisioned.

A discussion of future applications for ESIGN

The paper "Analysis of the SPV Secure Routing Protocol: Weaknesses and Lessons" [12] provides one example of the value and future application of ESIGN. For example, for the intercommunication of gateways that form the Internet, the Border Gateway Protocol (BGP) is employed. BGP is the protocol that binds tens-of-thousands of disparate Autonomous Systems (ASes) together. Within these ASes, the Secure Path Vector (SPV) protocol is employed to simultaneously provide strong routing authenticity, yet SPV must be high performance to manage the high volume of traffic [12].

Unfortunately, SPV has a weak m-times signature scheme, whereas SPV is only secure if the adversary A has less than m signatures. Various short hop/long hop schemes could be used by an adversary, A, to obtain m signatures, and to then forge signatures for routed information; *where the adversary, A, could then present the forged signatures as if the information is authentic*. Such an attack presents significant security risks. While numerous improvements to SVP have been contemplated, one obvious choice is to use the ESIGN scheme to provide a fast and secure version of the SPV security protocol within the BGP (*see* [12] at pg. 37]. ESIGN could be likewise used to implement other security related protocols.

In all, ESIGN when void of leakage and/or non-randomness/ bias, ESIGN provides EUA-CMA based on a sufficiently intractable AERP, thus ESIGN is an efficient and secure method for digital signatures and other future applications.

Bibliography

[1] R. Rivest, A. Shamir, and L. Adleman, "A Method for Obtaining Digital Signatures and Public Key Cryptosystems." Commun. ACM, 21(2) pgs. 120–126, 1978.

[2] Atsushi Fujioka, Tatsuaki Okamoto, Shoji Miyaguchi, "ESIGN: An Efficient Digital Signature Implementation for Smart Cards", Advances in Cryptology - EUROCRYPT '91, Volume 547 of the series Lecture Notes in Computer Science pp 446-457, 1991

[3] Tatsuaki Okamoto Eiichiro Fujisaki Hikaru Morita, "TSH-ESIGN: Efficient Digital Signature Scheme Using Trisection Size Hash (Submission to P1363a)", NTT Laboratories, 1998.

[4] Alfred Menezes, Minghua Qu, Doug Stinson, Yongge Wang, "Evaluation of Security Level of Cryptography: ESIGN Signature Scheme", Certicom Research, 2001

[5] Author Unknown, "Self Evaluation of ESIGN Signature", at link: security.nknu.edu.tw/crypto/Self_ESIGN.pdf, date unknown.

[6] Louis Granboulan, "How to repair ESIGN", Proceeding SCN'02, Proceedings of the 3rd international conference on Security in communication networks, pgs. 234-240, 2002, © 2003.

[7] Nick Howgrave-Graham, "A Review of the ESIGN digital standard", NTRU Cryptosystems at link: https://www.ipa.go.jp/security/enc/CRYPTREC/fy15/doc/1007_esign.pdf, date unknown.

[8] Dan Boneh and Hovav Shacham, "Fast Variants of RSA", CryptoBytes, Vol. 5, No. 1, pgs. 1-9, 2002.

[9] Tatsuaki Okamoto and Jacques Stern, "Almost Uniform Density of Power Residues and the Provable Security of ESIGN", C.S. Laih (Ed.): ASIACRYPT 2003, LNCS 2894, pp. 287–301, 2003.

[10] Pierre-Alain Fouque, Nick Howgrave-Graham, Gwenaelle Martinet, and Guillaume Poupard, "The Insecurity of Esign in Practical Implementations", C.S. Laih (Ed.): ASIACRYPT 2003, LNCS 2894, pp. 492–506, 2003.

[11] Noboru Kunihiro and Kaoru Kurosawa, "Deterministic Polynomial Time Equivalence between Factoring and Key-Recovery Attack on Takagi's RSA", 10th International Conference on Practice and Theory in Public-Key Cryptography Beijing, China, pp 412-425, 2007.

[12] Anton Mityagin, Saurabh Panjwani, and Barath Raghavan, "Public Review for Analysis of the SPV Secure Routing Protocol", Subtitle "Analysis of the SPV Secure Routing Protocol: Weaknesses and Lessons" ACM SIGCOMM Computer Communication Review 29 Volume 37, Number 2, pgs. 29-38, 2007

[13] Tetsutaro Kobayashi, Eiichiro Fujisaki, "Security of ESIGN-PSS", IEICE Transactions on Fundamentals, Vol. E90, No. 7, pgs. 1395-1405, 2007.

[14] Santanu Sarkar, "Small secret exponent attack on RSA variant with modulus $N = p^r q$", Des. Codes Cryptogr. 73:383–392, 2014.

[15] Bruce Schneier, "Applied Cryptography", John Wiley and Sons Publishing, 1994.

Appendix A – Glossary of Terms

AERP	Approximate e-th Root Problem
AER	Approximate e-th Root
AS	Autonomous System
BGP	Border Gateway Protocol
CMA	Chosen Message Attack, (also known as EUA-CMA)
ECDSA	Elliptic Curve Digital Signature Algorithm
ED25519	Edwards Curve 25519
ECF	Elliptic Curve Factoring
ECM	Elliptic Curve factoring Method
ESIGN	An efficient digital signature algorithm [2].
ESIGN-D	ESIGN-Deterministic
ESIGN-R	ESIGN-Randomized
ESIGN-PSS	ESIGN-Probabilistic Signature Scheme
ESIGN-PSS-R	ESIGN-Probabilistic Signature Scheme with Recovery
EUA-CMA	Existentially Unforgeable Against a Chosen Message Attack (also known as CMA)
GCD	Greatest Common Divisor
LLL	A lattice algorithm developed by A. K. Lenstra, H.W. Lenstra, Jr. and L. Lovasz, usually called, thus called the LLL algorithm.
NFS	Number Field Sieve
RSA	Rivest, Shamir, and Adelman
SVP	Shortest Vector Problem (relates to factoring)
SVP	Secure Path Vector (relates to BGP)

TSH	Tri-Section Hash
TSH-ESIGN	Tri-Section Hash-ESIGN