Performance and Security of ECDSA

Sharon Levy

***Abstract*—Digital signatures are used world wide to verify the authenticity of messages and confirm that they have not been altered in transmission. The Digital Signature Algorithm (DSA) is a Digital Signature Standard for the Federal Information Processing Standard and uses public key cryptography. The Elliptic Curve Digital Signature Algorithm (ECDSA) is a version of DSA using elliptic curves.**

In this paper, I will introduce ECDSA and discuss its key generation, signing, and verifying procedures. Then, I will compare this algorithm to the RSA digital signature algorithm and discuss its various advantages and drawbacks. Finally, I will discuss the security of ECDSA and attacks that can break it.

I. Introduction

Public key cryptography is a form of cryptography in which there exists a public and private key. In terms of digital signatures, the private key is used for creating signatures and the public key is copied and handed out to validate signatures. This has a huge advantage over secret or symmetric key cryptography since there is no need to find a secure way to swap keys[3].

The application of digital signatures can be executed across many different fields. It can be used within legal documents or to sign email messages. Digital signatures are used as a way to provide data authenticity, integrity and non-repudiation[7]. Essentially, when a digital signature is accepted, it ensures the receiver that the message was indeed sent by the source and was not altered in transmission. The Elliptic Curve Digital Signature Algorithm is a public key algorithm and was proposed by Scott Vanstone in 1992 as an alternative to the Digital Signature Algorithm[2]. This uses points on an elliptic curve as opposed to numbers in $\mathbb{Z}_p^*$[6]. ECDSA has been proven to be more effective than using DSA as it provides the same security with a smaller key size[9].

II. Algorithms

ECDSA consists of three algorithms: key generation, signing, and verification. The key generation algorithm computes a public and private key to use in the signing and verification processes. To create the actual digital signature, the signing procedure is executed. Finally, the verification method performs to prove the authenticity of the signature. The hash function used in the signing and verification algorithms is typically SHA-1[5]. The hardness of ECDSA comes from solving the Elliptic Curve Discrete Logarithm Problem (ECDLP). Below, I describe the three algorithms:

ECDSA Key Generation
Input: G and n where G is a base point on $E(\mathbb{F}_q)$ with q being equal to odd prime p or a power of 2 and n is the order of point G
Output: Public key Q and private key d

```
1: Randomly compute an integer $d$ in the interval [1,$n$ - 1]
2: Calculate $Q$ = $dG$
3: The public key computed is $Q$ and the private key is $d$
```

ECDSA Signature Generation
Input: d, G, n, hash function H, and message m
Output: Signature (r, s)

```
1: Compute a random integer $k$, within $1 \leq k \leq n$ - 1
2: Compute $kG = (x_1, y_1)$ and convert $x_1$ to an integer $\bar{x_1}$
3: Compute $r = \bar{x_1} \bmod n$. If $r$ = 0 then go to step 1
4: Compute $k^{-1} \bmod n$
5: Compute H($m$) and convert this bit string to an integer $e$
6: Compute $s = k^{-1}(e + dr) \bmod n$. If $s$ = 0 go
   to step 1
7: The signature for the message $m$ is $(r, s)$
```

ECDSA Signature Verification
Input: (r, s), m, n, e, G, Q, and hash function H
Output: Accept or reject signature (r, s)

```
1: Verify that $r$ and $s$ are integers in the interval [1, $n$ - 1]
2: Compute H($m$) and convert this bit string to an integer $e$
3: Compute $w = s^{-1} \bmod n$
4: Compute $u_1 = ew \bmod n$ and $u_2 = rw \bmod n$
5: Compute $X = u_1G + u_2Q$
6: If $X = \mathcal{O}$(the point at infinity), reject the signature.
   Otherwise, convert $x_1$ of $X$ to an integer $\bar{x_1}$ and
   compute $v = \bar{x_1} \bmod n$
7: Accept the signature if and only if $v = r$
```

III. ECDSA vs RSA

RSA is another public key cryptography algorithm. It was invented by Ron Rivest, Adi Shamir, and Leonard Adleman in 1977[1]. Unlike ECDSA, RSA can be used to encrypt and decrypt data in addition to verifying digital signatures. However, its encryption and signature algorithms include a hash function, similarly to ECDSA, where SHA-1 is also commonly used. The key generation algorithm of RSA is set up to produce a public and private key from the order of the product of two large prime integers. Therefore, the hardness of RSA, and the main method of breaking it, comes from solving the prime factorization problem[4]. Below, I describe the three algorithms:

RSA Key Generation
Input: None
Output: Public key (n, e) and private key d

```
1: Select two random prime numbers $p$ and $q$ with similar
```

```
   bit lengths
2: Compute $n = pq$
3: Compute $\Phi(n) = (p-1)(q-1)$
4: Compute $d$ where $ed = 1 \bmod \Phi(n)$ and $e$ is a random
   integer such that $e < \Phi(n)$ and $\gcd(e, \Phi(n)) = 1$
5: $(n, e)$ is the public key and $d$ is the private key
```

RSA Signature Generation
Input: d, n, hash function H, and message m
Output: Signature s

```
1: Compute $s = H(m)^d \pmod{n}$
2: $s$ is the signature generated
```

RSA Signature Verification
Input: s, e, n, m, and hash function H
Output: Accept or reject signature s

```
1: Compute $h = s^e \pmod{n}$
2: If $h = H(m)$, accept the signature
```

When comparing ECDSA to RSA, a major factor to discuss is key size. The following table shows the key lengths for RSA and ECDSA with the same level of security[10].

RSA Key Lengths (bits)	ECDSA Key Lengths (bits)
1024	192
2048	256

TABLE I

It is evident that ECDSA requires a much smaller key length in order to provide the same security as RSA. A major advantage of having this smaller key size is that computations can be executed faster. In addition, this helps reduce storage space, power consumption, processing power, and bandwidth[2].

In another study[1], the times for key generation, signature, and verification algorithms were computed with comparable key sizes for ECDSA and RSA. The results of the report (See tables 2, 3, and 4) showed that ECDSA outperformed RSA in both key and signature generations. However, RSA was able to verify messages much faster than ECDSA. The key sizes for ECDSA ranged from 163 to 571 bits and 1024 to 15360 bits for RSA algorithms. The times for ECDSA in the key generation were consistently faster than those of RSA. By the last comparison, RSA took a total of 679.06 seconds while ECDSA lasted 1.44 seconds, significantly faster. Meanwhile, the signature generation had slightly different results. RSA started out by executing faster than ECDSA. However, as the bit sizes for each increased, RSA was shown to slow down as ECDSA sped up and surpassed its counterpart on the final execution. Finally, with signature verification, RSAs times were considerably quicker than ECDSAs times and barely increased as the size of key lengths grew.

Thus, it appears that ECDSA has more advantages over RSA. Its small key sizes are beneficial in environments where resources such as storage space are limited. In addition, it runs its key and signature generation algorithms much faster than RSA. A scenario in which one would want to use RSA is when verifying more signatures than the number that are produced.

Key Generation			
Key Length (bits)		Time (secs)	
ECC	RSA	ECC	RSA
163	1024	0.08	0.16
233	2240	0.18	7.47
283	3072	0.27	9.80
409	7680	0.64	133.90
571	15360	1.44	679.06

TABLE II

Signature Generation			
Key Length (bits)		Time (secs)	
ECC	RSA	ECC	RSA
163	1024	0.15	0.01
233	2240	0.34	0.15
283	3072	0.59	0.21
409	7680	1.18	1.53
571	15360	3.07	9.20

TABLE III

Signature Verification			
Key Length (bits)		Time (secs)	
ECC	RSA	ECC	RSA
163	1024	0.23	0.01
233	2240	0.51	0.01
283	3072	0.86	0.01
409	7680	1.80	0.01
571	15360	4.53	0.03

TABLE IV

IV. Security of ECDSA

There are two attacks against digital signatures: a key-only attack in which the adversary only knows the public key and a message attack where the adversary has access to some signatures before cracking the function. There are several interpretations of what it means to break a digital signature: retrieving the secret key, creating another signing algorithm with an equivalent secret key, forging a signature for a chosen message, and forging a signature for at least one message[12].

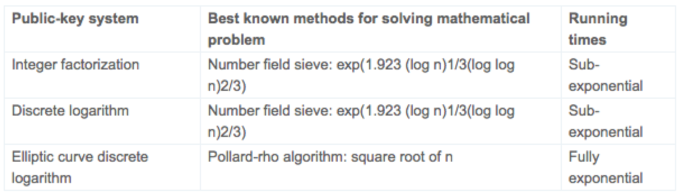
In order for ECDSA to be useful, it must have a high security (i.e. it is not easy to break). There are a few security conditions for ECDSA that are essential[5]:

• The discrete logarithm in the subgroup spanned by G is hard. This ensures that you cannot easily solve the discrete logarithm problem and therefore obtain the secret key.

• The hash function used is a one-way collision-resistant hash function. Being one-way means that you cannot determine m from $H(m) = y$. A collision-resistant function has a low probability of mapping two messages to the same thing (i.e. $H(m_1) = H(m_2)$).

• The generator for k is unpredictable. Without this, the secret key can be obtained using k, r, and s.

In the case of ECDSA, the two principal attacks on it are either against the hash function used within the signature generation or versus ECDLP[2]. Should the algorithm not incorporate the second bullet point from above, an adversary can find a collision within the hash function with two separate messages and sign one but declare his signature on the other. ECDLP is defined as solving for d in $Q = dG$ within the key generation algorithm. There are many known attacks against ECDLP including the exhaustive search, Pohlig-Hellman, and Baby-Step Giant-Step algorithms. One of these attacks is the Pollard's Rho algorithm, which has a running time of $(\sqrt{n\pi})/2$, where n is the order of point G. However, this algorithm can be parallelized and run on r different processors, so that the new running time is $(\sqrt{n\pi})/2r$.

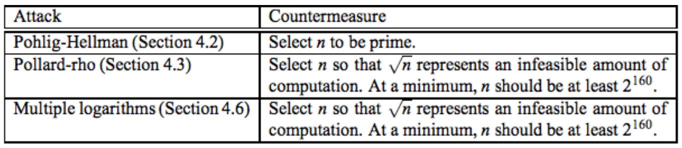


Public-key system	Best known methods for solving mathematical problem	Running times
Integer factorization	Number field sieve: exp(1.923 (log n)1/3(log log n)2/3)	Sub-exponential
Discrete logarithm	Number field sieve: exp(1.923 (log n)1/3(log log n)2/3)	Sub-exponential
Elliptic curve discrete logarithm	Pollard-rho algorithm: square root of n	Fully exponential

TABLE V

In table 5, a comparison is shown between the integer factorization, discrete logarithm, and elliptic curve discrete logarithm problems[11]. Each relates to RSA, DSA, and ECDSA, respectively. The best known algorithms for solving these problems are shown with their running times. For DSA and RSA the method given is number field sieve which is used for factorization. The process given for ECDSA is Pollard's Rho, though the parallelized version is actually faster. It is evident that ECDSA is more secure than DSA and RSA due to the fact that the running time to break it is fully exponential versus breaking DSA and RSA, which is only sub-exponential and therefore a lot faster.

The table below explains different methods in diverting various attacks against ECDLP, where the elliptic curve E is defined over $\mathbb{F}_q$. Variable n in this case is the order of point G and is assumed to be prime. The Multiple Logarithms approach is used to speed up attacks on ECDLP with the same elliptic curve parameters by using the solutions to previous ECDLP attacks[8].



Attack	Countermeasure
Pohlig-Hellman (Section 4.2)	Select n to be prime.
Pollard-rho (Section 4.3)	Select n so that $\sqrt{n}$ represents an infeasible amount of computation. At a minimum, n should be at least 2^{160}.
Multiple logarithms (Section 4.6)	Select n so that $\sqrt{n}$ represents an infeasible amount of computation. At a minimum, n should be at least 2^{160}.

TABLE VI

One flaw that has been pointed out in ECDSA is within the verification scheme. When r is checked against $\bar{x_1}$ mod n, it is evident that y_1 is not used at all in this comparison. Thus, there are now two signatures that will be valid for a given message: (r, s) and $(r, -s \bmod n)$[5].

Other attacks that can be used against ECDSA relate to the security of k within the signature generation algorithm. After each message is signed, k must be destroyed since an adversary can compute the secret key d using $d = r^{-1}(ks - e)$ mod n. Similarly, k must be regenerated for different messages. If the attacker knows that two messages have the same k, he can recover the secret key d from the two signatures (r, s_1) and (r, s_2). This is done by seeing that $ks_1 \equiv e_1 + dr (\text{mod } n)$ and $ks_2 \equiv e_2 + dr \pmod{n}$. Using this information, you can subtract ks_2 from ks_1 and get $ks_1 - ks_2 \equiv e_1 - e_2 \pmod{n}$. Dividing both sides by $s_1 - s_2$ gives us $k \equiv (e_1 - e_2)(s_1 - s_2)^{-1} \pmod{n}$. With this knowledge of k, an adversary can recover the secret key using the attack described above[2].

V. Conclusion

ECDSA has been shown to be a better alternative to both RSA and DSA for producing digital signatures. I compared the three ECDSA algorithms of key generation, signing, and verification to those of RSA. The results produced showed that ECDSA excelled with its running time in both key generation and signing but failed in verification against RSA.

The main benefits of ECDSA include the smaller key sizes that achieve the same security, making it useful when being implemented in hardware, and the hardness of breaking ECDLP, which is incorporated into the algorithm. Though there are attacks against ECDSA, like the Pollard's Rho and Pohlig-Hellman algorithms, they have running times that are much slower than those against RSA and DSA. The given requirements for ECDSA relating to the hash function, discrete logarithm, and number generator ensure that the statements above are true.

References

[1] Arrendondo, Brandon and Jansma, Nicholas. ”Performance Comparison of Elliptic Curve and RSA Digital Signatures”, 2004. Web. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.129.7139&rep=rep1&type=pdf

[2] Johnson, Don and Menezes, Alfred. ”The Elliptic Curve Digital Signature Algorithm (ECDSA)”, 1999. Web. http://cacr.uwaterloo.ca/techreports/1999/corr99-34.pdf

[3] Blumenthal, Matt. ”Encryption: Strengths and Weaknesses of Public-key Cryptography”, Web. http://www.csc.villanova.edu/~tway/courses/csc3990/f2007/csrs2007/01-pp1-7-MattBlumenthal.pdf

[4] Goyal, Vikas and Kaur, Amanpreet. ”A Comparative Analysis of ECDSA v/s RSA Algorithm”, 2013. Web. http://www.interscience.in/IJCSI_Vol3Iss1/23-25.pdf

[5] Vaudenay, Serge. ”The Security of DSA and ECDSA”, 2002. Web. https://www.iacr.org/archive/pkc2003/25670309/25670309.pdf

[6] Stern, Jacques. ”Evaluation Report on the ECDSA signature scheme”, Web. https://www.ipa.go.jp/security/enc/CRYPTREC/fy15/doc/1004_R3_ECDSA.pdf

[7] Hankerson, D., Menezes, A., Scott, A. *Guide to Elliptic Curve Cryptography*. New York: Springer 2003. Print.

[8] Menezes, Alfred. ”Evaluation of Security Level of Cryptography: The Elliptic Curve Discrete Logarithm Problem (ECDLP)”, 2001. Web. https://www.ipa.go.jp/security/enc/CRYPTREC/fy15/doc/1028_ecdlp.pdf

[9] Khalique, A., Singh, K., Sood, S.”Implementation of Elliptic Curve Digital Signature Algorithm”, May 2010. Web. http://www.ijcaonline.org/volume2/number2/pxc387876.pdf

[10] Ali, Al Imem. ”Comparison and Evaluation of Digital Signature Schemes Employed in NDN Network”, June 2015. Web. http://arxiv.org/pdf/1508.00184.pdf

[11] Vanstone, S.A. ”Next Generation Security for Wireless: Elliptic Curve Cryptography”, 2003. Web. http://www.sciencedirect.com/science/article/pii/S0167404803005078

[12] Goldwasser, S., Micali, S., Rivest, R. ”A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks”, 1988. Web. https://people.csail.mit.edu/silvio/Selected%20Scientific%20Papers/Digital%20Signatures/A_Digital_Signature_Scheme_Secure_Against_Adaptive_Chosen-Message_Attack.pdf