A Framework for Security of DNS using Cryptography

Naveen Kumar
Research Scholar
Department of Computer Science and Engineering
Ganga Institute Of Technology & Management, Kablana

Kamal Kumar Ranga
Assistant Professor
Department of Computer Science and Engineering
Ganga Institute Of Technology & Management, Kablana

Abstract

DNS, Domain Name System is a protocol that resolves hostnames to IP Addresses over the Internet. DNS, being an open source, it is less secure and it has no means of determining whether domain name data comes from an authorized domain owner. So, these vulnerabilities lead to a number of attacks, such as, cache poisoning, cache spoofing etc. Hence, there is a need of securing DNS. Digital Signatures are a good way of authenticating the domain owners. The digital signatures generated with public key algorithms have the advantage that anyone having the public key can verify them. Existing proposals include public key cryptographic algorithms (e.g., RSA, DSA etc.) for securing DNS. With the technology growing faster everyone accesses internet through mobile phones whether it is used to check E-Mails or visiting any secure sites, ECDSA involving ECC (Elliptic Curve Cryptography) concepts having less key sizes as compared to RSA can be implemented to provide security to DNS.
Keywords: DNS, ECDSA, cryptographic algorithms

I. INTRODUCTION

The Domain Name System is a protocol for locating domain names and mapping them to IP addresses. DNS is a hierarchical, distributed database, which provides mapping between easy to remember hostnames, such as www.mdurohtak.ac.in, and IPv4 or IPv6 network addresses, for example, 117.211.115.134. When a hostname is translated into its numeric representation, this allows the network to trace a path from a user to a particular server. Correct and timely DNS translations are vital for networks such as the Internet and thus are an interesting target for attackers.

Fig. 1: Basic DNS functionality

II. METHODOLOGY USED

The elliptic curve digital signature algorithm is the elliptic curve analogue of DSA and serves the same purposes of key generation, signature generation, and signature verification. ECDSA was first proposed in 1992 by Scott Vanstone in response to NIST's proposal of DSS. It was later accepted in 1998 as an ISO standard (ISO 14888-3), as an ANSI standard (ANSI X9.62) in 1999, and as an IEEE standard (IEEE 1363-2000) and as a NIST standard (FIPS 186-2) in 2000.

A. Algorithm

1) Key Pair Generation

Let A be the signatory for a message M. Entity A performs the following steps to generate a public and private key:

1) Select an elliptic curve E defined over a finite field Fp such that the number of points in E (Fp) is divisible by a large prime n.
2) Select a base point, P, of order n such that $P \in E(Fp)$
3) Select a unique and unpredictable integer, d, in the interval [1, n-1]
4) Compute Q = dP
5) Sender A's private key is d
6) Sender A's public key is the combination (E, P, n, Q)

2) Signature Generation

Using A's private key, A generates the signature for message M using the following steps:

1) Select a random number k to be used only once, that is, for every new signature generation of a message, a new k is selected, such that $1 < k < n-1$
2) Generate (r, s) component of signature such that

- k.G = (x, y)
- r = x modulo n
- if r = 0 then repeat 2 again

3) Calculate hash of message (M) whose signature is to be generated, i.e., e = h(M)
4) $s = k^{-1}(e + (d*r))$ modulo n

3) Signature Verification

The receiver B can verify the authenticity of A's signature (r, s) for message M by performing the following:

1) Obtain signatory A's public key (E, P, n, Q)
2) Verify that values r and s are in the interval [1,n-1]
3) Calculate $u1 = e*s^{-1}$ modulo n
4) Calculate $u2 = r*s^{-1}$ modulo n
5) Calculate T = u1.G + u2.Q = (x1, y1), where '.' is point multiplication and '+' is point addition and can be calculated using elliptic curve arithmetic.
6) Calculate v = x1 modulo n
7) The signature for message M is verified only if v = r

B. Security of ECDSA

The generation of the public key in ECDSA involves computing the point, Q, where Q = dP. In order to crack the elliptic curve key, attacker would have to discover the secret key d. Given that the order of the curve E is a prime number n, then computing d given dP and P would take roughly $2^{n/2}$ operations. For example, if the key length n is 192 bits (the smallest key size that NIST recommends for curves defined over GF(p)), then attacker will be required to compute about 2^{96} operations that takes around two and a half trillion years to find the secret key.

Also, given a point R = kP, where R and P are known, then there is no way to find out what the value of 'k' is. Since, there is no point subtraction or point division, to resolve k = R/P.

This thing where multiplicand can't be found even when the original and destination points are known is the whole basis of the security behind the ECDSA algorithm, and the principle is called a trap door function or Elliptic Curve Digital Logarithmic Problem (ECDLP).

C. Modified ECDSA

1) Signature Generation

Using A's private key, A generates the signature for message M using the following steps:

1) Select a random number k to be used only once, that is, for every new signature generation of a message, a new k is selected, such that 1 < k < n-1
2) Generate (r, s) component of signature such that

- k.G = (x, y)
- r = x modulo n
- if r = 0 then repeat 2 again

3) Calculate hash of message (M) whose signature is to be generated, i.e., e = h(M)
4) $s = d(r*k - e)^{-1}$ modulo n //changed formula

2) Signature Verification

The receiver B can verify the authenticity of A's signature (r, s) for message M by performing the following:

1) Obtain signatory A's public key (E, P, n, Q)
2) Verify that values r and s are in the interval [1,n-1]
3) Calculate $u1 = e*r^{-1}$ modulo n //changed formula
4) Calculate $u2 = (r*s)^{-1}$ modulo n //changed formula
5) Calculate T = u1.G + u2.Q = (x1, y1), where '.' is point multiplication and '+' is point addition and can be calculated using elliptic curve arithmetic.
6) Calculate v = x1 modulo n
7) The signature for message M is verified only if v = r

D. Comparison between RSA and ECDSA

Parameters	*RSA*	*ECDSA*
Key Size (same security)	*1024 bit length Bigger*	*192 bit length Smaller*
Signature Generation	*Slow*	*Fast*
Signature Verification	*Fast*	*Slow*

Encryption	*Fast*	*Slow*
Decryption	*Slow*	*Fast*
Key Exchange	*Slow*	*Slow*

III. IMPLEMENTATION

The purpose of using a Java applet is to provide a familiar and easily accessible medium for users to sign and verify messages using the elliptic curve digital signature algorithm. By using a Java applet, our implementation can be embedded into the software and made available for authoritative zones.

The ECDSA applet contains three parts:

1) IP Address generation
2) Key generation
3) Signature generation
4) Signature verification.

The user interface for showing simulation of algorithm is show below, followed by an outline of typical usage of the applet.

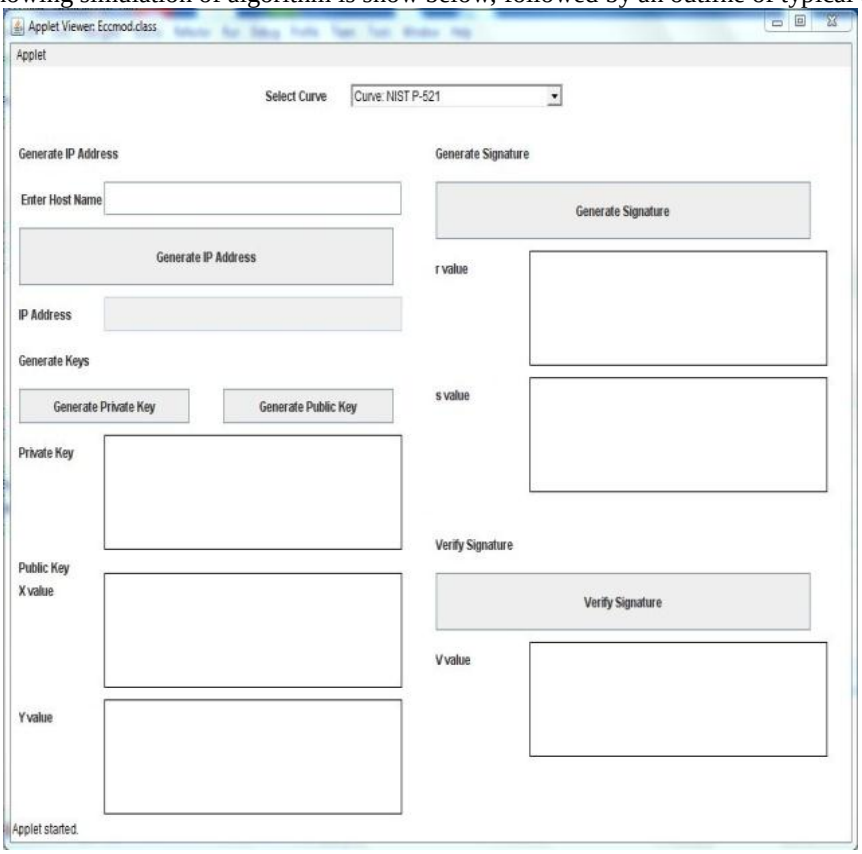

Fig. 2: ECDSA Applet

The ECDSA Applet is composed of one main class: the ECDSA class. This class implements the applet and performs all of the arithmetic computations and ECDSA functionality that the user requests when signing an IP Address on the applet or verifying a signature. The user can select from a list of curves with various key sizes to sign their address. Following is a summary of the functions and packages used to implement the ECDSA class.

A. Global Curve Values

The parameters for the NIST elliptic curves over GF (p) can be found in Appendix A. These parameters include the order r, base point coordinates (x, y) (where x, y $\in$ GF (p) and the point is of order r), the n-bit prime modulus p, the coefficient b, and the coefficient a (which always has a value of –3 for efficiency purposes). All of these parameters satisfy the equation $y^2 = x^3 + ax + b$. These parameters were hard-coded into the applet. They were then put into an array so that when the user selected a particular curve from the drop-down list in the applet, all of selected curve's parameters can be easily referenced.

B. Applet User Interface and Components

All of the applet components are first instantiated. The applet is then initialized in the init() method. The init() method initializes all of the components by calling initComponents(). The sizes and locations of all the applet components are set in this

method and then added to the applet. The last thing that the initComponents() method does is to add an ActionListener to each button in the applet. It also adds an ItemListener to the drop down list of NIST curves to use.

C. ItemListeners and ActionListeners

The ItemListen and ActionListen classes implement the ItemListener and ActionListener classes, respectively. These two classes handle all events in the applet that deal with components. All button presses and list selections are handled differently and the listeners that were added to each component will call these classes to properly deal with an event when it happens.

When a user selects a curve to use in signature generation or verification, the ItemListen class will receive the selected index in the drop down list. This translates to the array index where the curve parameters of the selected curve are stored. A new elliptic curve is created using the EllipticCurve class with these curve parameters taken from the arrays. The ActionListen class handles all other component events and is much longer. It has only one method – actionPerformed() – that specifies different actions for all events.

D. The actionPerformed() Method

The following is a breakdown of how the actionPerformed() method handles each event.

Table- Summary of handling applet events

Generate IP Address button pressed	*A code runs that uses network connection to generate IP Address of the entered host name.*
Generate Private Key button pressed	*A random big integer is generated that will serve as the private key for the user. The Java SecureRandom class is used to create the key. Modulo of the order is performed on the private key. The private key of the selected curve size is, then, displayed in the text field.*
Generate Public Key button pressed	*The public key is created using the multPoint() method with the user's private key and base point as parameters. The base points (x, y) are, then, displayed in their respective text fields.*
Generate Signature button pressed	*The applet calls the sign() method and generates a signature for the IP Address. The signature is then displayed in the text fields.*
Verify Signature button pressed	*The verify() method is called to verify the signature using the public key. The v value of the signature is shown and a message is displayed stating whether the signature is valid or invalid.*

E. Additional Elliptic Curve Classes

Two elliptic curve classes were created within the main ECDSA class. They are the ECPoint class and the EllipticCurve class. These classes provide the ability to create an elliptic curve point and elliptic curve, respectively, as well as the ability to easily access these values. It should be noted that the functionality of both of these classes are available in Java SDK 2 version 1.5. Our system was constrained to use of Java v.1.6.

F. ECPoint Class

Our ECPoint class has the same methods as the Java class but the constructor is slightly different. A third value was added to indicate if the point was at infinity. The equals() method also compares the point to another ECPoint instead of an object. The following details the ECPoint methods.

Table - ECPoint class constructor summary

Constructor Summary
ECPoint(BigInteger x, BigInteger y) *Creates an ECPoint from the specified affine x-coordinate x and affine y-coordinate y.*

Table - ECPoint class constructor summary

Method Summary	
boolean	*equals(ECPoint pt)* *Compares this elliptic curve point for equality with the specified point.*
BigInteger	*getAffineX()* *Returns the affine x-coordinate x.*
BigInteger	*getAffineY()* *Returns the affine y-coordinate y.*

G. EllipticCurve Class

Our EllipticCurve class differs from the Java one in that the constructor doesn't take a third field argument. Below is the EllipticCurve class methods summary.

Table- EllipticCurve class constructor summary

Constructor Summary
EllipticCurve(BigInteger a, BigInteger b) *Creates an elliptic curve with the coefficients a and b.*

Table- EllipticCurve class method summaries

Method Summary	
BigInteger	*getA()* *Returns the first coefficient a of the elliptic curve.*
BigInteger	*getB()* *Returns the second coefficient b of the elliptic curve.*

H. Complexity Comparison of ECDSA and Variant ECDSA

According to the paper given by Hu Junru [15], the complexity of original ECDSA and variant ECDSA (I.E. 1, I.E. 2), given in Appendix B, is shown in table below along with the ECDSA (ECDSA v1) implemented in my thesis work.

Table- ECDSA Comparison

Algorithm	*Point Doubling* $O(n^2 \log n)$	*Point multiplication* $O(n^2)$	*Inverse* $O(n^2)$	*Total*
ECDSA	*Sign*	*1*	*1*	$(\log n + 11)n^2$
	Verify	*2*	*1*	$(2\log n+11)\, n^2$
I.E. 1	*Sign*	*1*	*1*	$(2\log n+3)n^2$
	Verify	*2*	*1*	$(2\log n+11)n^2$
I.E. 2	*Sign*	*1*	*1*	$(\log n + 11)n^2$
	Verify	*2*	*0*	$(2\log n+2)n^2$
ECDSA v1	*Sign*	*1*	*1*	$(\log n+2)n^2$
	Verify	*2*	*2*	$(2\log n+4)n^2$

IV. CONCLUSION

There are various security measures adopted in DNS using public key cryptography, which includes RSA and DSA. With the technology growing day by day, there is a need of same level of security with smaller key sizes. Now, everyone uses mobile to retrieve data from internet and mobile being small and portable device needs security with less power consumption. This can be done with the help of ECC by implementing ECDSA in DNS.

Also, nowadays everyone uses their smart phones to extract contents from the Internet. Whether phones are used for opening various websites, receiving emails, filling up online forms etc., operating these huge RSA secured web content is time and memory consuming both. So, there is a need of faster verifier on these small handheld devices to authenticate the web sources quickly and with less power and memory consumption. The function of quick verification with small bit sizes of keys used is given by ECDSA.

REFERENCES

Book:

[1] William Stallings, Cryptography and Network Security, 4th Edition, Pearson Education, Inc., 2011
[2] Neetesh Saxena, Narendra S. Chaudhari, "Secure Encryption with Digital Signature Approach for Short Message Service", IEEE, 2012.
[3] Suranjith Ariyapperuma and Chris J. Mitchell, "Security vulnerabilities in DNS and DNSSEC", IEEE Computer Society
[4] Aqeel Khalique Kuldip Singh Sandeep Sood, "Implementation of Elliptic Curve Digital Signature Algorithm", International Journal of Computer Applications (0975 – 8887), Volume 2 – No.2, May 2010.
[5] Vivek Kapoor, Vivek Sonny Abraham, Ramesh Singh, "Elliptic Curve Cryptography", May 20-26, 2008. ACM Ubiquity, Volume 9, Issue 20.
[6] Nils Gura, Arun Patel, Arvinderpal Wander, Hans Eberle, Sheueling Chang Shantz, "Comparing Elliptic Curve Cryptography and RSA on 8-bit CPUs".
[7] Ramaswamy Chandramouli and Scott Rose, "Challenges in securing the domain name system", US National Institute of Standards and Technology, The IEEE Computer Society, 2006.
[8] Giuseppe Ateniese , Stefan Mangard , "A New Approach to DNS Security (DNSSEC)".
[9] D. Sravana Kumar, CH. Suneetha, A. Chandrasekhar, "Encryption of Data using Elliptic Curve over Finite Fields", International Journal of Distributed and Parallel Systems (IJDPS) Vol.3 (January 2012), No.1.
[10] Ghanmy Nabil, Khlif Naziha,, Fourati Lamia, Kamoun Lotfi, "Hardware implementation of Elliptic Curve Digital Signature Algorithm (ECDSA) on Koblitz Curves", 8th IEEE, IET International Symposium on Communication Systems, Networks and Digital Signal Processing.
[11] Kadjo Tanon Lambert, Oumtanaga Souleymane, Kone Tiemoman, Abba Brice, and Tety Pierre, "Deployment of DNSSEC: Problems and Outlines".
[12] Muhammad Yasir Malik, "Efficient Implementation of Elliptic Curve Cryptography Using Low-power Digital Signal Processor".
[13] Sachin Kumar Sinha, Avinash Kant Singh, Amaresh Sharma, "Security System for DNS using Cryptography".
[14] Tarun Narayan Shankar, G. Sahoo(April / May 2009), "Cryptography with Elliptic Curves", International Journal Of Computer Science And Applications Vol. 2, No. 1.

[15] Hu Junru, "The Improved Elliptic Curve Digital Signature Algorithm", International Conference on Electronic & Mechanical Engineering and Information Technology, 2011
[16] Antonio Lioy, Fabio Maino, Marius Marian, Daniele Mazzocchi, "DNS Security". Terena Networking Conference, May 22-25, 2000, Italy.
[17] Joe Gersch and Dan Massey, "Deploying DNS Security (DNSSEC) in Large-Scale Operational Environments", Cybersecurity Applications & Technology Conference For Homeland Security.
[18] M.Prabu, Dr. R.Shanmugalakshmi, "A Comparative Analysis of Signature Schemes in a New Approach to Variant on ECDSA", International Conference on Information and Multimedia Technology, 2009.
[19] Features of OpenDNSSEC. Available: http://www.opendnssec.org/features/
[20] ECC Tutorial. Available: http://certicom.com/index.php/ecc-tutorial
[21] Recommended Elliptic Curves For Federal Government Use, July 1999. Available: http://csrc.nist.gov
[22] Domain Name Security Extensions (DNSSEC). Available: https://www.menandmice.com/resources/articles/dnssec/
[23] BIND, Internet Systems Consortium. Available: http://www.isc.org/products/BIND/