World Academy of Science, Engineering and Technology
International Journal of Computer and Information Engineering
Vol:15, No:4, 2021

Alternative Key Exchange Algorithm Based on Elliptic Curve Digital Signature Algorithm Certificate and Usage in Applications

A. Andreasyan, C. Connors

Abstract**—**The Elliptic Curve Digital Signature algorithm-based X509v3 certificates are becoming more popular due to their short public and private key sizes. Moreover, these certificates can be stored in Internet of Things (IoT) devices, with limited resources, using less memory and transmitted in network security protocols, such as Internet Key Exchange (IKE), Transport Layer Security (TLS) and Secure Shell (SSH) with less bandwidth. The proposed method gives another advantage, in that it increases the performance of the above-mentioned protocols in terms of key exchange by saving one scalar multiplication operation.

Keywords**—**Cryptography, elliptic curve digital signature algorithm, key exchange, network security protocols.

I. Introduction

IN this article, an alternative method of key exchange algorithm is proposed based on using Elliptic Curve Digital Signature (ECDSA) X.509 certificate algorithm parameters. The proposed key exchange algorithm idea is based on using ephemeral and static ECDSA public keys. All current well-known protocols (IKE, TLS, SSH) use the Elliptic Curve Diffie-Hellman (ECDH) algorithm for key exchange, which is based on four scalar multiplication operations. Two scalar multiplication operations are done on both the initiator and responder sides. This method allows to speed up the key exchange handshake by saving one scalar multiplication operation. The proposed method can be used in applications where secure connection establishment is critical. This article is not intended to design a new protocol, but the proposed method can be incorporated with any currently known protocols, such as IKE, TLS, SSH and those based on ECDSA X.509 certificate.

Another advantage of the proposed method is that it can be used in the load balancer with an ECDSA X.509 certificate on the server side. This method eliminates the creation of separate tunnels from the load balancer to server or cluster, as it is done today.

II. ECDH Key Exchange Based Network Security Protocols

The DSA (Digital Signature Algorithm) and ECDSA (Elliptic Curve Digital Signature Algorithm) algorithms are well known and NIST (National Institute of Standard) approved algorithms. These algorithms are widely used in all network security protocols, such as IKEv1/v2, SSL/TLS and SSH.

All publicly known security key exchange protocols are based either on DH, ECDH or RSA algorithm. For example, IKEv1/v2 based on DH and ECDH algorithm; SSL/TLS on RSA, DH, ECDH; SSH on DH, ECDH. Usually, DSA and ECDSA algorithms are used for authentication purposes only. All above mentioned network security protocols use ECDSA based X.509 [1] certificate to authenticate peer only. As an example, let us consider a TLS client server secure connection from the point of view of DH or ECDH key exchange only [2], [3]. In TLS protocol, the server authentication is mandatory and client authentication is optional. Let us assume that the client and server both have X.509 ECDSA type of certificates and client authentication is mandatory also. In order for the client and server to create a secure connection, the client authenticates the server first by verifying server's X.509 certificate authenticity. The shared secret key is generated using either DH or ECDH algorithms. Then, the server authenticates the client and generates the same DH or ECDH shared secret key. Both sides use a shared secret to generate data record's confidentiality and authentication keys.

The above described key exchange algorithm applies to IKE [4] and SSH [5] protocols as well.

The ECDH algorithm uses scalar multiplication operation for generating an ECDH ephemeral public key and a shared secret. This operation is very resource and time-consuming, specifically in devises with low computational resources.

In ECDH key exchange algorithms, two scalar multiplication operations are required on each side [2]. In the ECDHE_ECDSA cipher spec case, the server generates and sends ephemeral ECDH public key (point) to the client along with its certificate and ECDH domain parameters. The client uses domain parameters and generates its ephemeral public key (point) and shared secret key by doing two scalar multiplication operations. Then it sends its public point to the server along with its certificate. The server does the same operation and generates a shared secret. Overall, four scalar multiplication operations are required to generate the shared secret, two on the server side and two on client side. This cipher spec is commonly used in TLS protocol [2].

The above described ECDHE_ECDSA cipher spec is represented in the steps below. This method uses different domain parameters for the base point from the server and client certificates. The prime order is different from ECDSA

A. Andreasyan and C. Connors are with VMware Inc. 3429 Hillview Ave Palo Alto, CA 94304 USA (e-mail: aandreasyan@vmware.com, cconnors@vmware.com).

parameters; will be marked as G` and p`. First step: Alice generates random integer *x(A)*from [2, p-1] and computes Alice EC DH public key – $Y_A = X_A * G'$. X_A is EC DH private key. Alice signs and sends Y_A to Bob along with her CERT(A)

REQUEST [{CERT(A), Y(A)}] Sig
----------→

Second step: By receiving REQUEST Bob verifies REQUEST signature and computes Bob DH public key $Y_B = X_B * G`$ where $X_B$ is Bob ECDH private key randomly generated from [2, p-1]. Then Bob computes the shared secret point by multiplying Alice ECDH public key $Y_A$ with ECDH private key $X_B$ $Z = Y_A * X_B$ Since $Y_A = X_A * G`$ then $Z= X_A * X_B * G`$ Bob signs REPLY and sends CERT(B) with EC DH public key *Y(B)* to Alice.

REPLY [CERT(B), Y_B] Sig
←----------

Third step: By receiving REPLY Alice computes the shared secret point by multiplying Bob's EC DH public key with her own EC DH private key *x(A).*$Z= Y_B * X_A$. Since $Y_B = X_B * G`$ then shared secret point is $Z= X_A * X_B * G'$

After these steps, both sides generate a Z shared secret, which requires four scalar multiplication operations.

With low computational resources, devices are very critical to speed up the key exchange process. There are already proposed methods to solve this problem. For example:

- Precompute DH or ECDH public keys and keep them in public key pool. Later, these keys are used during secure connection establishment time. This method reduces connection establishment time but still takes computational resources, since the public key pool should be updated periodically.
- Another approach is to use a statically generated DH public key from X.509 certificate. This approach is part of TLS protocol [1]. In the ECDH_ECDSA key exchange, the ECDSA certificate contains an ECDH public key and the server does not generate a public key separately. The server only sends its certificate to the client. The client authenticates the server. Then it generates an ECDH public key using the server X.509 certificate EC domain parameters and the server's ECDH public key from the X.509 ECDSA certificate. Then it generates the shared secret key. As we see, the client does two scalar multiplication operations. The client sends its public key and X509 certificate to the server. The server authenticates the client by verifying the certificate and using the client's ECDH public key to generate a shared secret. Overall, three scalar multiplication operations are required to obtain a shared secret: two scalar multiplications on the client side and one on the server side. The disadvantage of this cipher spec is that it does not provide perfect forward secrecy, since the server ECDH public is fixed in the certificate, and for this reason, it is not commonly used in practice.

The proposed method tries to solve this problem in a different way. In an elliptic curve domain, the scalar multiplication is the most time and resource consuming operation and saving one scalar multiplication will speed up the whole key exchange process by 25%.

III. Key Exchange Based on X.509 ECDSA Certificate Parameters

To reduce the number of scalar multiplications, the proposed algorithm combines the ECDH and ECDSA domain parameters. For the sake of simplicity, let us consider that the client and server have elliptic curve domain parameters ECDH_ECDSA, which are part of the X509 certificate.

The proposed key exchange method can work with any type of curves and provides perfect forward secrecy. This method incorporates ECDSA X.509 certificate parameters with the ECDH key exchange.

Let us assume that Alice (client) and Bob (server) have P-curve ECDSA type of certificates. These types of certificates have the following parameters T(*p, r, b, G*), where *p*- prime module, *r*- order, *b* EC coefficient, and *G* base point, which are common for Alice and Bob [1].

Each certificate has a Q public key and corresponding d private key. The public and private key generation method is out of the scope of this article. These keys will be used only for signature generation and verification and not for key exchange.

The proposed key exchange method uses only a G base (generator) point, prime order, and the peer's (server) ECDSA public key. It is a two pass - REQUEST/REPLY protocol, after which, both peers have the same shared secret. Alice has X509 CERT(A) ={*p, r, b, G,* Q_A }, where d_A is an ECDSA private key and $Q_A = d_A * G$ is ECDSA public key Bob X509 CERT(B) ={*p, r, b, G,* Q_B d_B is an ECDSA private key $Q_B = d_B * G$ is ECDSA public key .

In the first step, Alice generates random integer X_A from [2, r-1] and computes a one-time public key - $Y_A = X_A * G$. X_A is unique for each session. Alice signs the request and sends it to Bob along with her CERT(A).

REQUEST[{CERT(A), Y_A}] Sig
----------→

In the second step, Bob verifies the REQUEST signature and computes the shared secret point by multiplying the one-time public key Y_A with the ECDSA private key X_B $Z = Y_A * d_B$. Since, $Y_A = X_A * G$ then $Z= X_A * G * d_B$. Bob sends REPLY with CERT(B) to Alice.

REPLY [CERT(B)]
←----------

In the third step: by receiving REPLY, Alice computes the shared secret point by extracting Bob's ECDSA public key from certificate CERT(B) and multiplying previously, randomly generated X_A. $Z = Q_B * X_A$ and since, $Y_B = X_B * G$. then the shared secret point is $Z = d_B * G * X_A$.

After these steps, both sides have the same shared secret key Z, which can be used for generating data records confidentiality and authentication keys. The proposed method requires three scalar multiplication operations, where the regular ECDH case four scalar multiplication operations are required. Since each new connection uses new generated ephemeral ECDH keys, it therefore provides perfect forward secrecy.

Comparing the proposed method with the ECDHE_ECDSA cipher spec, it saves one scalar multiplication and provides shorter payload length as well.

In applications, where a client connects to known servers, then the servers X.509 certificates must be known before starting a new connection. This situation allows skipping the REPLY step from the client and the key exchange can be done in one step. The REPLY step scalar multiplication operation can be combined in the REQUEST step operation and the shared point can be calculated right after generating a random integer and calculating $Y_A = X_A * G$, *and* $Z = d_B * G * X_A$. The proposed approach is very similar to the RSA type of key exchange based on the RSA type of X.509 certificate used in TLS protocol.

In the above description, we considered that both sides have common ECDSA parameters, which assumes that Alice's and Bob's certificates are issued by the same CA. The proposed method can be used with different ECDSA parameters as well. In this case, the server certificate must be available before the client initiates a connection with server.

The proposed method allows speeding up the key exchange process by saving one scalar multiplication operation. The proposed method is not an ECDH_ECDSA cipher spec replacement. This method can be used in some applications where the secure connection establishment process is critical.

IV. Load Balancer with X.509 ECDSA Certificate.

In current technology, SSL traffic often needs to be decrypted on a load balancer before passing a request. This is called SSL-termination. This approach saves a server or cluster of servers from spending time on decrypting traffic. It also allows consolidating network-based services, such as web application firewalls, intrusion detection and compression as well as caching in one single point. However, it results in security concerns since traffic between the load balancer and server is not encrypted.

Another approach is to pass SSL traffic (called pass-through) to a server, which then does the decryption. This however requires more CPU powers on the server side. This covers the security concern but requires creating a secure tunnel between server and load balancer.

To cover the security concern and avoid creating extra tunnels between the load balancer and server, the server's X.509 certificate and private key can be loaded into the load balancer. The load balancer can decrypt the traffic and apply all security services and then pass the client's encrypted traffic to the server. This mechanism works only when a server has the RSA type of certificate. When a server has either ECDSA or DSA types of certificates, the load balancer needs to create additional tunnels between the server and clusters. Creating additional secure tunnels is a resource heavy and time-consuming operation. It requires managing and reestablishing these tunnels due to timeout also.

The proposed method can be well applied in this scenario and avoid creating additional tunnels between the load balancer and server or clusters. If a client operates in an aforementioned way, the servers ECDSA or DSA certificate can be loaded into the load balancer, which allows it to generate the same shared secret key along with the server or clusters. Later on, the shared secret key can be used to decrypt the client's traffic on the load balancer without creating additional tunnels with the server or cluster. The server or cluster will use the same shared secret to decrypt the client's traffic as well.

In this application, the load balancer allows to save CPU time on the server or clusters sides and have a centralized point to apply deep packet inspection policy. This approach is similar with the RSA type of certificate approach and allows the load balancer to operate in the same way for the server certificate.

V. Mathematical Proof of the Proposed Algorithm

Let us assume that Alice (client) and Bob (server) have P-curve ECDSA type of certificates. This type of certificate has the following parameters T(*p, r, a, b, G*), where *p*- prime module, *r*- order, a and *b*- EC coefficient (field elements) [6], *G*- base point, which are common for Alice and Bob.

The keys for the ECDSA are computed as follows:

- Use an elliptic curve E with
  - module p
  - coefficients a and b (where can be a=0)
  - a point G which generates a cyclic group of prime order r
- Choose a random integer d with $0 < d < r$
- Compute $Y = d*G$

The keys are:

- Public Key = (p, r, a, b, Y, G). The ECDSA certificate contains all these parameters,
- Private Key = d

Alice and Bob both have certificates issued from the same root of trust. Alice's certificate -Cert(A) contains parameters (p, r, a, b, Y_A, G) and Bob's certificate Cert(B)-(p, r, a, b, Y_B, G).

When Alice ties to exchange a key with Bob, the following steps must be done:

Step1.

- generate random integer - one-time private key X_A from [1, r-1]
- compute one-time public key (point multiplication) $Y_A = X_A * G$
- send one-time public key - Y_A along with her x.509v3 certificate -Cert(A) to Bob

Step2. By receiving Y_A and Cert(A) Bob does the following operations:

- computes (point multiplication) shared secret key $Z = Y_A * d_B$

- sends his x.509v3 certificate - Cert(B) to Alice

Step3. By receiving Bob's certificate, Alice does the following operation:

- computes a shared secret key Z, using Bob's Y_B public key from certificate and her one-time private key X_A - $Z= Y_B * X_A$

Let us prove that the Z-shared secret key is the same on both sides.

From the above described ESDSA key generation, Alice and Bob have the following public keys (p, r, a, b, Y_A, G) and (p, r ,a, b, Y_B, G), respectively, where $Y_A = d_A * G$ and $Y_B = d_B * G$ where $1 < d_A < r-1$, $1 < d_B < r-1$ and G is base point of prime order of r. The p is prime field-GF(p), where all point arithmetic is implemented in terms of modulo p. The elements of this filed are integers of modulo p. The field can be binary $GF(2^m)$ as well. In this case point arithmetic will be done on binary bit string. The point arithmetic type does not affect algorithm validity.

To prove the proposed algorithm validity, let us use the following theorem. We will not prove the theorem but will use cyclic group property of the theorem.

Theorem*:* The points on an elliptic curve (E) together with in an identity (or natural) element have cyclic subgroups E(GF(p)). Under certain conditions all points in an elliptic curve can form a cyclic group [7].

Based on cyclic group properties, a primitive element must exist, such that its power generates entire group. Let's denote G primitive element (or base point as mentioned above) of order r, where r is prime number.

Alice generates $1 < X_A < r-1$, since G is a primitive element with order r then $Y_A = X_A * G$ (point multiplication) will result in another point from the same cyclic group.

Bob computes $Z = Y_A * d_B$, and based on the same cyclic group property, it will result in point Z from the same cyclic group, so Z and Y_A belongs to the same cyclic group. Since $Y_A = X_A * G$, let's replace Y_A to get $Z=X_A * G * d_B$. Now let us prove that Bob's computed shared secret-Z is the same as what Alice computes by receiving Bob' certificate. Alice computes $Y_B * X_A$, since $Y_B=d_B * G$ by replacing Y_B, to get $Z=d_B * G * X_A$. Based on cyclic subgroup multiplicative property $Z=X_A * G * d_B = d_B * G * X_A$. As we see both sides obtain the same shared secret key-Z.

VI. Conclusion

The proposed method allows ECDSA x509v3 certificates to be used not only for authentication purposes, but it opens new domain of usage such as in load balancers, where only one session can be created between load balancer and web server. It also saves one scalar multiplication operation in key exchange, which saves resources and bandwidth specifically for IoT devices.

References

[1] Federal Information Processing Standards (FIPS) 186-4, Digital Signature Standard, 2013

[2] T. Dierks E. Rescorla The Transport Layer Security (TLS) Protocol Version 1.2, 2008

[3] E. Rescorla The Transport Layer Security (TLS) Protocol Version 1.3, 2018

[4] C. Kaufman, P.Hoffman, Y.Nir, P. Eronen T. Kivinen Internet Key Exchange Protocol Version 2 (IKE), 2014

[5] T. Friedl, N. Provos, W. Simpson Deffie-Hellman Group Exchange for Secure Shell Transport Layer Protocol, 2006

[6] L. Bassam, D Johnson, W. Polk Internet X509 Public Key Infrastructure 1999pp 5

[7] Cristof Paar, Jan Pelzl Understanding Cryptography, 2010, pp 246