Team 5

Professor Chen

CS.4600.02

23 March 2023

ECDSA and Bitcoin Applications

Elliptic Curve Digital Signature Algorithm or ECDSA, is a cryptography algorithm with the purpose of ensuring the data transfer between two parties is unaltered and of authentic origin. ECDSA creates digital signatures based on the files hashes so receivers can verify using the public key available to them.

The algorithm utilizes an elliptic curve cryptography or ECC. A basic elliptical curve equation being: $y^2 = (x^3 + ax + b) \bmod p$. The value, p is a large prime number and the upper limit for a randomly generated private key ranging from 0 to p-1, while a and b are parameters for the elliptic curve.

First, ECDSA will generate a random elliptic curve based on government and industry standards such as secp256r1, secp256k1, or secp384r1. In Bitcoin, the curve secp256k1 is used, along with the parameters a = 0 and b =7. This elliptical curve and its parameters will be shared between users for them to be able to calculate and verify values. Private key, dA, will be randomly generated from numbers 0 to p-1, and using the point of generator G of prime order on the curve, we can calculate the public key. Q_a, the public key, is found via dA * G. The private key is kept secret, and is combined with the file hash to generate the digital signature, while the public key will be transmitted along with the curve parameters and files for receivers to verify.

1. $Q_a = dA \times G$

For example, to generate a 160-bit digital signature, which is made up of two 80-bit halves - R and S. A point on the curve, point P needs to be found in order to calculate the two halves. Point P will be calculated from k×G, with k being a random 80-bit value generated from 0 to p-1. R is the x-coordinate value of P. For S, the message that is being sent needs to first be hashed. Using a hash algorithm, currently SHA256 or better, message *M* is hashed to find the 80-bit value Z, and we can compute S using the equation $k^{-1}(z + R * dA) \bmod p$. With R and S found, the signature will be sent alongside the message to the recipient who will then use their own public key and the parameters to verify the signature.

2. $P = k*G$

3. $Z = SHA256(M)$

4. $S = k^{-1}(z + R * dA) \bmod p$

The receiver will verify the signature by calculating the same point P on the curve that the sender had found. This can be achieved by using the shared public key, and other shared values like a, b, G, p, R, and S. Using the equation $(S^{-1} * z * G) + (S^{-1} * R * Qa)$, the point P can be calculated. To verify the signature, the value R must match with the x-coordinate of the point P. Otherwise, the signature is invalid and we can't be sure if the files are tampered with or came from authentic origin.

5. $P = (S^{-1} * z * G) + (S^{-1} * R * Qa)$

ECDSA is an effective and secure algorithm due to its astronomical number of possible parameters yet easy to verify. Elliptical curves can be extremely expansive, though for the algorithm it is finite, meaning that it is almost impossible to find a way of duplicating the signature without the private key. Given the shape of elliptical curves, you can only determine point values through one-way functions utilizing the curve. This makes reverse engineering

values extremely difficult without knowing the private key or exploiting developers' carelessness. Point addition and multiplication methods are one-way or trapdoor functions. For example of point addition, if there are points P and Q are on an elliptical curve you can form a line that passes through them and intersects with point -R. The inverse of -R is point R which is also the sum of P + Q. R can only be found through this one-way function, meaning you cannot determine P or Q even if you were given R. Point multiplication takes this a step further through repeated addition with k * P. The random value k times P can produce the same result but in a scalar manner, allowing for values of greater irreversible complexity. One-way functions are like a padlock and key, to change the lock from open to closed is easy but to reverse that action requires a key or a great deal of bruteforce. That's why ECDSA is one of the more secure methods of modern cryptography. You can share smaller sized keys that cannot be used against you to calculate your own private key.

Since ECDSA is such a secure and lightweight algorithm, its implementation is becoming more widespread though RSA still tops the chart due to its longer history. ECDSA usage is still growing, used in important documents such as trusted certificates, securing communication between government bodies, and more widely known in cryptocurrency. In the cryptocurrency Bitcoin, ECDSA is the main method of guaranteeing the identity of traders and authenticity of funds being transferred through digital signatures. Signatures created by ECDSA will be sent packaged with the funds, allowing traders to verify identities and secure their money. Another added benefit for Bitcoin is addresses. In Bitcoin, each wallet has its own public address which is used to send and receive funds. The address is derived from the public key, which itself is created from using ECDSA on a private key. This address adds an extra layer of security, in the

event where private keys can be derived from public keys, as the address is being shared instead of the public key.

ECDSA can have vulnerabilities if not implemented correctly such as an issue with an older version of a Java-implemented ECDSA. This older version did not check R and S to ensure they were non-zero values, and so if both were 0, any entity could enter a blank signature which would be accepted. This vulnerability has since been patched in April 2022 by Oracle. A famous example is the 2010 Sony PS3 Hack when the digital signature was cracked due to the value *k* not being properly randomized by Sony developers. The vulnerability allows people to reverse-engineer the private key based on the generated return value by the verifying algorithm.

A possible attack avenue on ECDSA is through hashing collisions. By generating a number, j, that allows R = Q + j * G, an entity can find z = Rj mod p and set S = R. This allows the entity to input messages into the hash function until z = SHA256(*M*). This attack allows a hacker to create a valid signature without knowing the keys, but would take a tremendous amount of effort given the number of possible points on a curve.

Despite the possibilities of such attacks, ECDSA's complexity and security with proper implementation provides an effective method to check the integrity of data. Its numerous benefits of smaller key sizes, less resource consumption, and secured one-way algorithm makes it the highly desirable encryption method.

Bibliography

Barski, Conrad, and Chris Wilmer. "Cryptographic Methods Used in Bitcoin." *Bitcoin for the Befuddled*. *Ebary.net*, ebrary.net/7941/education/signing_bitcoin_transaction_using_ecdsa. Accessed 20 Mar. 2023.

Buchanan, Bill. "Not Playing Randomly: The Sony PS3 and Bitcoin Crypto Hacks." *ASecuritySite: When Bob Met Alice*, Medium, 12 Nov. 2018, medium.com/asecuritysite-when-bob-met-alice/not-playing-randomly-the-sony-ps3-and-bitcoin-crypto-hacks-c1fe92bea9bc. Accessed 20 Mar. 2023.

Centieiro, Henrique. "ECDSA — How to Programmatically Sign a Transaction." *Level Up Coding*, Medium, 20 June 2021, levelup.gitconnected.com/ecdsa-how-to-programmatically-sign-a-transaction-95eec854bca7. Accessed 20 Mar. 2023.

Certicom Research. *SEC 2: Recommended Elliptic Curve Domain Parameters*. 27 Jan. 2010. *Standards for Efficient Cryptography*, www.secg.org/sec2-v2.pdf. Accessed 20 Mar. 2023.

Corbellini, Andrea. "ECDSA.py." 14 Dec. 2015. ECDSA Python Implementation.

Digital Signature Standard (DSS). US Department of Commerce National Institute of Standards and Technology, July 2013. *National Institute of Standards and Technology*, https://doi.org/10.6028/NIST.FIPS.186-5. Accessed 20 Mar. 2023.

"Elliptic Curve Signatures." *Saylor Academy*, learn.saylor.org/mod/book/tool/print/index.php?id=36341. Accessed 20 Mar. 2023.

Froehlich, Andrew. "Elliptical Curve Cryptography." *TechTarget*, 4 Jan. 2022, www.techtarget.com/searchsecurity/definition/elliptical-curve-cryptography. Accessed 20 Mar. 2023.

Levy, Sharon. *Performance and Security of ECDSA*. 2015. *CSU San Bernardino*, koclab.cs.ucsb.edu/teaching/ecc/project/2015Projects/Levy.pdf. Accessed 20 Mar. 2023.

Marshall, Blair. Weblog post. *How Does ECDSA Work in Bitcoin*, Medium, 22 Feb. 2018, medium.com/@blairlmarshall/how-does-ecdsa-work-in-bitcoin-7819d201a3ec. Accessed 20 Mar. 2023.

Rykwalder, Eric. "The Math Behind the Bitcoin Protocol." *Coindesk*, 11 Sept. 2021. *CoinDesk*, www.coindesk.com/markets/2014/10/19/the-math-behind-the-bitcoin-protocol/. Accessed 20 Mar. 2023.

Snifikino. "Understanding How ECDSA Protects Your Data." *Instructables*, Autodesk, www.instructables.com/Understanding-how-ECDSA-protects-your-data/. Accessed 20 Mar. 2023.

Teske, Edlyn. Weblog post. *Explaining the Java ECDSA Critical Vulnerability*, Cryptomathic, 3 May 2022, www.cryptomathic.com/news-events/blog/explaining-the-java-ecdsa-critical-vulnerability. Accessed 20 Mar. 2023.

Walker, Greg. *ECDSA | How to Create Public Keys and Signatures in Bitcoin*. 26 Aug. 2021. *Learn Me a Bitcoin*, learnmeabitcoin.com/technical/ecdsa. Accessed 20 Mar. 2023.

"Why Digital Signatures Are Essential for Blockchains." *Coinbase*, 25 Jan. 2022, www.coinbase.com/cloud/discover/dev-foundations/digital-signatures. Accessed 20 Mar. 2023.