HW1: ECDSA Malleability and Bitcoin

CSE528: Introduction to Blockchain and Cryptocurrency

Mihir Chaturvedi, 2019061

Q1. Why is ECDSA malleable?

ECDSA Generation

- **E**lliptic **C**urve **D**igital **S**ignature **A**lgorithm is a method to ensure the authenticity of the signed data
- Signature generation via ECDSA produces a pair of values: R and S
- A temporary key pair is created for generating the R and S values, from a **random** seed number k
  - Private key: k
  - Public key: $P = k * G$
    - G - Generator point, a constant point on the elliptic curve in all transactions
    - Here, as per multiplication on elliptic curves, P is a point on the elliptic curve (specifically `secp256k1` for Bitcoin) represented by an x- and y-coordinate.
- Now,
  - R = x-coordinate of point P
  - $S = k^{-1} * (m + A * R) \bmod n$
    - k - temporary private key
    - m - (hash of the) message/data
    - A - signing private key (belonging to the user)
    - R - x-coordinate of point P
    - n - prime order of the elliptic curve

ECDSA Verification

- To verify the authenticity of a signed message, that is, ensure that the received values *R* and *S* have been generated by signing the data using only the signer's private signing key, we perform calculations in an attempt to **retrieve the point *P*** on the elliptic curve, and **compare its x-coordinate with *R***.
- To retrieve *P*:
  - $P = S^{-1} * (m*G + R*Q)$
    - *R* and *S* - Signature values
    - *m* - (hash of the) message/data that was signed
    - *Q* - signing public key (belonging to the user)
    - *G* - Generating point of the elliptic curve
- Finally, to verify we compare the x-coordinate of this *P* with the input value *R*. If they are equal, it is ensured that this signature was signed by only the signer's private key.
- Nota bene: the Signer's private key is not used to verify authenticity

ECDSA's Inbuilt Malleability

- The generated pair *R* and *S* are not authoritative - it is possible for a **different value of *S* to pass the verification scheme**. In other words, ***(R, S)* is a malleable signature**
- Reason:
  - Verification is positive as long as *R* is equal to the x-coordinate of the retrieved point *P*
  - A different value of *S*, say *S'*, that yields the same x-coordinate of *P* (in essence, *R*) will pass the verification
  - Since *P* is on an elliptic curve, there exists a point on the curve with the same x-coordinate as *P*, but a negated y-coordinate: ***-P***
    - $-P = -(k*G) = (-k)*G$
    - From the equation to generate *S*, we can deduce that *-k* can be obtained by simply negating *S*.
  - **Thus, *-S mod n* will yield *R***
    - *n* - the prime order of the curve
- Consequence:
  - Programs that use the ECDSA signature *(R, S)* as an authoritative identifier for the signed data are susceptible to face problems
  - Two different signatures can be used the verify the same data, thereby introducing ambiguity

Q2. What is the issue of using malleable ECDSA in Bitcoin?

Transaction Malleability

- Bitcoin transactions make use of the transaction ID, `txid`, as the identifier of a transaction
    - `txid` is constructed using the *R* and *S* ECDSA signature pair generated by supplying it with the hashed data of the transaction
- As explained in the previous section, since an easily computable and different value of *S* can produce a valid *R*, correspondingly, a different `txid` can too be constructed.
- Consequently, **one transaction can have two valid and identifying `txid`s**.
- Thus, **until the transaction has been mined and committed** to the chain, a transaction's `txid` can be changed/modified and is **not authoritative**.

Malleability Attacks

- Transaction malleability exposes an attack vector by which users conducting transactions can be *tricked* into falsely believing their initial transaction was unsuccessful, and thus repeat their transaction.
- Transaction malleability does not compromise the security or architecture of the Bitcoin blockchain, but rather introduces a nuisance conducted by bad actors in the network.
- Demonstration:
    - Alice initiates a transaction destined to Bob's address with txid = T_1.
    - A bad-actor node intercepts the transaction, modifies the txid by simply changing the *S* value to -*S mod n* and reconstructing txid.
        - The new txid = T_2
    - Since the signature is still valid, it is mined in a block and committed to the blockchain. The transaction is complete.
    - But, Alice's wallet uses txid=T_1 to track the status of the transaction, and since no transaction with txid=T_1 was mined and pushed into the blockchain for some time, it declares it as an unsuccessful transaction.
    - Unwittingly, Alice creates another transaction destined to Bob in hope that this transaction shall succeed.
        - Alice has been "attacked", and the malleability has caused her to double-spend

Q3. How could the Bitcoin blockchain overcome the issue?

Bitcoin's Workaround

- Bitcoin works around this flaw in ECDSA by enforcing a canonical representation of the ECDSA signature:
    - When generating the transaction signature, both values of the possible S are calculated: S and $-S \bmod n$
    - The lower of the two S values (aka "low-S") is chosen as the canonical representation, and is used to create `txid`
        - low-S is guaranteed to be lower than $n/2$, where n is the prime order of the curve
        - Consequently, high-S is greater than $n/2$
    - While relaying or mining the transaction, if its `txid` is greater than $n/2$, the transaction is rejected.
- This enforcement was originally to be introduced in BIP62 in March 2014, but was finally enforced by merging PR #6769 into Bitcoin Core in October 2015.
    - low-S checking is only done if the opcode `SCRIPT_VERIFY_LOW_S` is present in the transaction script, which by default is present in all recent Bitcoin implementations.
- Further, in BIP141, Segregated Witness (Segwit) was introduced into the Bitcoin protocol which eliminated the dependence on ECDSA-constructed `txid` by introducing a new, non-malleable type of transaction ID called "witness txid" or `wtxid`.

References

- Mastering Bitcoin: Programming the Open Blockchain, 2nd Edition by Andreas M. Antonopoulos
- https://yondon.blog/2019/01/01/how-not-to-use-ecdsa/
- https://github.com/bitcoin/bips/blob/master/bip-0062.mediawiki
- https://github.com/bitcoin/bitcoin/pull/6769
- https://eklitzke.org/bitcoin-transaction-malleability