Response to the question "what is the basis of SECDSA security?" posted on

https://crypto.stackexchange.com/questions/110997/

blinding-an-ecdsa-private-key-without-learning-the-private-key

Eric R. Verheul

eric.verheul@keycontrols.nl

13th March 2024

The security of Split-ECDSA (SECDSA) [2,3] is based on the security of *raw* ECDSA signing, i.e. not letting the cryptographic hardware compute the hash value itself but instead but letting it sign only the result, i.e. the *hash value*. A hash value is simply a byte array representing a big integer of the size of the hash output, e.g. 32 bytes in the case of SHA256 use. In the raw signing context, the hash is computed by the application calling the cryptographic library/hardware. As cryptographic hardware typically has computational or communicational restraints, raw signing is typically used in practice. This is why most (if not all) cryptographic libraries/hardware including the iOS/Secure Enclave, Android-/HBK+Strongbox, TPMs, PKCS11 based HSMs support this.

Raw ECDSA signing security means that the following attack is not possible: an attacker requests the cryptographic hardware to sign a series of chosen hashvalues $H_1, H_2, ..., H_n$ with the ECDSA private key u (and public key $u{\cdot}G$) and is then able to generate a signature based on ECDSA private key u on a hashvalue H that was not requested by the attacker. One can argue that raw ECDSA signing security is commonly accepted to hold, as raw signing is commonly used and supported by most (if not all) cryptographic libraries/hardware.

We can base SECDSA raw signing security on raw ECDSA signing security of the underlying possession key u (with public key $u{\cdot}G$). That is, we can show that if SECDSA allows for an attack as indicated above, one can also find one for ECDSA itself which we argued is not possible.

We reason as follows. First of all, from the proof of Proposition 3.1 [2] (most specifically the sequence of equalities appearing in it) one can conclude the following (actually the converse from what is used in Proposition 3.1):

If (r, s) is an ECDSA signature on hash value e with private key $u{\cdot}\sigma$ (and public key $\sigma{\cdot}u{\cdot}G$) then $(r, s{\cdot}\sigma^{-1})$ is an ECDSA signature on hash value $e{\cdot}\sigma^{-1}$ with private key u.

Now suppose that an attacker can break SECDSA security, i.e. the attacker can call Algorithm 6 of [2] to SECDSA sign specific hashvalues $H'_1, H'_2, ..., H'_n$ leading to a SECDSA signature (r, s) on a hashvalue H' not requested. From the above

conclusion it then follows that $(r, s{\cdot}\sigma^{-1})$ is signature on hashvalue $H'{\cdot}\sigma^{-1}$ with private key u. Also, in Algorithm 6 of [2], the calls for signing with private u are $H'_1{\cdot}\sigma^{-1}, H'_2{\cdot}\sigma^{-1}, ..., H'^{-1}_n{\cdot}\sigma$ which are all different from $H'{\cdot}\sigma^{-1}$ as $H'_1, H'_2, ..., H'_n$ and H' are different. We have arrived at an successful attack on raw ECDSA signing security with the underlying possession key u

The use of SECDSA deriving blinded keys from one hardware backed private key u is not explicitly mentioned in the SECDSA paper [2]. It is explained in this Linkedin post [4]. In this post it also explained how one can use a Schnorr Zero-Knowledge Proof (or simply a Schnorr or ECDSA signature) to prove to verifiers (attestation issuers and relying parties in the EUDI-wallet context) that blinded keys are based on the same possession key u. It is also indicated that this technique can be used in the presentation of multiple attribute attestations whereby proving in a privacy friendly way to a verifier that all the private attestation keys are bound to the same hardware and person. This technique is thereby a more secure and privacy friendly alternative for the "claim-based binding" technique of OpenID for Verifiable Credentials (OpenID4VC) [5].

The SECDSA paper [2] is from 2021, just at the time the European Digital Identity (EUDI) Wallet [1] was introduced. Although the SECDSA techniques can be applied to the EUDI-wallet, the paper focusses on qualified remote signing with an application to authentication at eIDAS assurance level High. The user is deployed a mobile application (APP) for (qualified) signing or authentication which is comparable to the EUDI-wallet avant la lettre. The user signing keys are managed by a remote signing server whereby the mobile signingapplication/user sends key management instructions to the server (generate key, sign with key etc.). One of the big issues of remote signing is that the server can sign on behalf of the user without the user has instructed the server. SECSA aims to achieve classical 'sole control'/non-repudiation on the instructions to the server: each server key management instruction from the APP/user is equipped with a APP/user SECDSA signature that achieves 'sole control'/non-repudiation. This means that any dispute on a server key management instruction can be conveniently handled by the SECDSA signed instruction. Apart from 'sole control'/non-repudiation, the SECDSA setup also aims that the authentication based on the SECDSA signed instruction achieves eIDAS assurance level High. This is rather subtle and requires that the server gets no information enabling him to bruteforce the PIN even with rooted access to the wallet. In [3] a compact explanation of this SECDSA usage is presented.

1 References

1. European Digital Identity Framework, European Parliament legislative resolution of 29 February 2024. See https://www.europarl.europa.eu/doceo/document/TA-9-2024-0117_EN.pdf.
2. https://eprint.iacr.org/2021/910.
3. https://www.cs.ru.nl/E.Verheul/presentations/SECDSAbasedEUDI-wallets.pdf.

4. https://www.linkedin.com/pulse/cryptographically-linking-edi-wallet-attribute-mobile-verheul-/
5. https://openid.net/sg/openid4vc/