WabiSabi wallet crypto audit

We reviewed the cryptographic algorithms in https://github.com/zkSNACKs/WalletWasabi/tree/master/WalletWasabi/Crypto for security defects (incorrect implementation, software bugs, randomness issues, data leaks, etc.).

The most complex part was the zero-knowledge module that implements generalised sigma protocols over the secp256k1 elliptic curve [SEC 2]. Given public inputs $\left(Y_1, \ldots, Y_m\right) \in \mathbb{G}^m$ and generators $\left(\{G_{1,i}\}_{i=1}^n, \ldots, \{G_{m,i}\}_{i=1}^n\right) \in \mathbb{G}^{m \times n}$, the prover can generate a proof that they know a witness $\{w_i\}_{i=1}^n \in \mathbb{F}^n$ such that

$$Y_j = \sum_{i=1}^{n} w_i G_{j,i}, \quad \text{for all } j = 1, \ldots, m$$

Considering $m = n = 1$, the above equation is equivalent to the discrete logarithm problem over the chosen group. These protocols are made non-interactive using the Strobe protocol [Ham17] to maintain a state from the transcript and "hashing" values out of it. This introduces a computation assumption relating to the security of the underlying hash function.

By carefully defining these relations, this construction can be used for range proofs (proof that a committed positive integer belongs to the interval $[0, 2^b - 1]$ for some predetermined upper bound on the number of bits b) and validation of WabiSabi credentials defined in [FKOS21].

As reference, we compared the implementation with a similar construction from [KO18] and found no discrepancies.

Specific issues and recommendations

- In ProofSystem.cs, typo line 169: it should be `rb_i -> r_i * b_i`
- In `CredentialIssuerSecretKey()`, we recommend to check that scalars are non-zero, as it would be an insecure key and would reveal other problems (with the PRNG).
- In the algebraic MAC in Mac.cs, a given `t` must not be repeated with the same key `sk`, otherwise part of the key leaks. The application seems to only call it with random values.
- The `GetInt()` implementation used in https://github.com/zkSNACKs/WalletWasabi/blob/58dbe7572df8386560b224906eec2bf009d1ca8f/WalletWasabi/Crypto/Randomness/WasabiRandom.cs#L31 must return uniformly distributed values (the actual implementation was unclear to us).
- In NBitcoin, `Scalar` elements constructed from a byte array can in certain cases overflow the modulus `p` of the scalar field of the curve. For this reason,

the constructor can additionally return a flag which indicated whether a reduction modulo `p` occurs. This flag is not checked in SyntheticSecretNonceProvider.Sequence() or `Transcript.GenerateChallenge()` and therefore may cause the comparisons against 0 to fail in `Equation.Verify()` and `Equation.Respond()`. While these events only happen with negligible probability, we recommend that the functions responsible for generating random scalars generate retry if the modulus is overflowed.

- We also noticed a typo in SchnorrBlindingSignature.cs#L80 where the wrong variable was being checked against zero. Instead of `_v.IsZero` it should be `_w.IsZero`.