Suppose that you received n

signatures, S_1

$\dots S_n$

, where each signature S_i

is itself an aggregate signature with pubkeys $P_{i,j}$

and messages $M_{i,j}$

for $1 \le j \le m_i$

. That is, $e(S_i, G) = \prod_{j=1}^{m_i} e(P_{i,j}, M_{i,j})$

.

The following is a technique for optimizing verification of all n

signatures at the same time, especially in the case where the same message is signed by many public keys (this is the reality in eth2).

Generate a list of n

random values between 1 and the curve order, r_1

$\dots r_n$

. Locally calculate $S^* = S_1 * r_1 + S_2 * r_2 + ... + S_n * r_n$

(writing the elliptic curve group additively), and $M'_{i,j} = M_{i,j} * r_i$

. Now, simply check that $e(S^*, G) = \prod_{i=1}^n \prod_{j=1}^{m_i} e(P_{i,j}, M'_{i,j})$

. If it is, then all signatures are with very high probability valid, and if it is not, then at least one signature is invalid, though we lose the ability to tell which one.

This reduces the number of final exponentiations we need to make to only one per block, and reduces the number of Miller loops from $n + \sum m_i$

to $1 + \sum m_i$

, at the cost of a few extra message multiplications.

Why this works

The signatures are a set of equations. It's an elementary algebraic fact that if a set of equations $y_1 = f(x_1)$

$\dots y_n = f(x_n)$

holds true, than any linear combination of those equations $y_1 * r_1 + ... + y_n * r_n = f(x_1) * r_1 + ... + f(x_n) * r_n$

also holds true. This gives completeness (ie. if the signatures are all valid, the check will pass).

Now, we need to look at soundness. Suppose that any of the signatures are incorrect, and specifically signature S_i

deviates from the "correct" signature C_i

by D_i

(ie. $D_i = S_i - C_i$

). Then, in your final check, that component of the equation will deviate from the "correct" value by $D_i * r_i$

, which is unknown to the attacker, because r_i

is unknown to the attacker. Hence, the attacker cannot come up with values for any of the other signatures that "compensate" for the error.

This also shows why the randomizing factors are necessary: otherwise, an attacker could make a bad block where if the correct signatures are C_1

and C_2

, the attacker sets the signatures to $C_1 + D$

and $C_2 - D$

for some deviation D

. A full signature check would interpret this as an invalid block, but a partial check would not.

Why not just make a big aggregate signature?

Why not just have blocks contain S^*

as a big aggregate signature of all messages in the block? The answer here is that it makes accountability harder: if any of the signatures is invalid, one would need to verify the entire block to tell that this is the case, so verifying slashing messages would require many more pairings.

This technique lets us get the main benefit of making a big aggregate signature for each block, leading to large savings in signature verification time, while keeping the ability to split off individual signatures for individual verification for accountability purposes.

Optimizations

We can gain some efficiency and sacrifice nothing by setting $r_1 = 1$

, removing the need to make multiplications in the signature that contains the most distinct messages. We can also set other r_i

values in a smaller range, eg. $1...2^{64}$

, keeping the cost of attacking the scheme extremely high (as the r_i

values are secret, there’s no computational way for the attacker to try all possibilities and see which one works); this would reduce the cost of multiplications by ~4x.