Reference

Thursday, May 2, 2024 11:09 PM

1. https://www.rareskills.io/post/groth16

Notes

Thursday, June 20, 2024 10:32 AM

Q1: What if we don't use alpha and beta?

Verifier:

$$\text{pairing}([A]_1, [B]_2) = \text{pairing}([C]_1, G_2)$$

these are cryptic coordinates in verifier's eyes

Verifier has no way to know if prover actually followed the protocol and provided him correct [A] [B] and [C]. It is very easy for the prover to forge a fake proof, for example:

$$\text{pairing}(3G_1, 5G_2) = \text{pairing}(15G_1, G_2)$$

```
ret2basic@PwnieIsland: ~ 80x24
>>> from py_ecc.bn128 import G1, G2, multiply, pairing
>>> term1 = multiply(G1, 3)
>>> term2 = multiply(G2, 5)
>>> term3 = multiply(G1, 15)
>>> term4 = G2
>>> pairing(term2, term1) == pairing(term4, term3)
True
>>>
```

In this case prover can just come up with 3, 5, and 15 without doing any computation.

A1: Introducing alpha and beta

α, β: random numbers generated by trusted setup

Trusted setup publishes $[\alpha]_1$ and $[\beta]_2$ (random shifts)

$$[A]_1 = [A_{old}]_1 + [\alpha]_1$$
$$[B]_2 = [B_{old}]_2 + [\beta]_2$$

Now attacker can't forge a fake proof using the method we just mentioned. Why? Because if he wants to find an equality a * b = c in the scalar field, he must know alpha and beta. But, the security of alpha and beta is guaranteed by ECDLP.

Let's walk through a failed hacking attempt (not rigorous, just let you see the idea):

Say hacker wants to come up with $a \cdot b = c$ s.t.

$$\text{pairing}(aG_1, bG_2) = \text{pairing}(cG_1, G_2) + \text{pairing}(\alpha G_1, \beta G_2)$$

Why? will discuss later

here $a = a' + \alpha$

$b = b' + \beta$

Attacker needs to find a' and b':

$$a \cdot b = c + \alpha\beta$$
$$(a' + \alpha)(b' + \beta) = c + \alpha\beta$$
$$a'b' + a'\beta + \alpha b' + \alpha\beta = c + \alpha\beta$$

unknown to attacker

This is "Attack 1: xxx" "Attack 2: xxx" and "Attack 3: xxx" in ZK book Groth16 chapter, but presented in a clearer way (personal opinion).

Attacker can:

1. Come up with a' and b', try to compute c -> need alpha and beta
2. Come up with c, try to compute a' and b' -> need alpha and beta
3. Come up with a and b, try to compute c -> need alpha times beta

➢ Conclusion: can't leak alpha, beta, also alpha times beta

After introducing alpha and beta, we QAP equation also changes:

old QAP

$$\sum_{i=0}^{m} a_i u_i(x) \sum_{i=0}^{m} a_i v_i(x) = \sum_{i=0}^{m} a_i w_i(x) + h(x)t(x)$$

$$(\alpha + \sum_{i=0}^{m} a_i u_i(x))(\beta + \sum_{i=0}^{m} a_i v_i(x))$$

$$= \alpha\beta + \beta \sum_{i=0}^{m} a_i u_i(x) + \alpha \sum_{i=0}^{m} a_i v_i(x) + \underbrace{\sum_{i=0}^{m} a_i u_i(x) \sum_{i=0}^{m} a_i v_i(x)}_{\text{substitution}}$$

$$= \alpha\beta + \beta \sum_{i=0}^{m} a_i u_i(x) + \alpha \sum_{i=0}^{m} a_i v_i(x) + \sum_{i=0}^{m} a_i w_i(x) + h(x)t(x)$$

Combine

substitution

$$\Rightarrow \underbrace{(\alpha + \sum_{i=0}^{m} a_i u_i(x))}_{A} \underbrace{(\beta + \sum_{i=0}^{m} a_i v_i(x))}_{B} = \boxed{\alpha\beta} + \underbrace{\sum_{i=0}^{m} a_i(\beta u_i(x) + \alpha v_i(x) + w_i(x)) + h(x)t(x)}_{C}$$

EC addition, and evaluate at τ

$$\Rightarrow [A]_1 = [\alpha + \sum_{i=0}^{m} a_i u_i(x)]_1 \Rightarrow [\alpha]_1 + \sum_{i=0}^{m} a_i [u_i(\tau)]_1$$

$$[B]_2 = [\beta + \sum_{i=0}^{m} a_i v_i(x)]_2 \Rightarrow [\beta]_2 + \sum_{i=0}^{m} a_i [v_i(\tau)]_2$$

$$[C]_1 = [\sum_{i=0}^{m} a_i(\beta u_i(x) + \alpha v_i(x) + w_i(x)) + h(x)(t(x))]$$

$$\Rightarrow \sum_{i=0}^{m} a_i [\beta u_i(\tau) + \alpha v_i(\tau) + w_i(\tau)]_1 + [h(\tau)t(\tau)]_1$$

___ means "computation is supported by trusted setup"

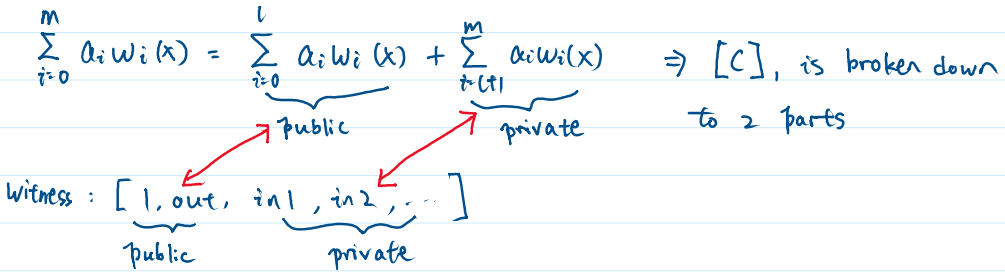
Prover computes $\text{proof} = ([A]_1, [B]_2, [C]_1)$ and sends to verifier.

Verifier now verifies:

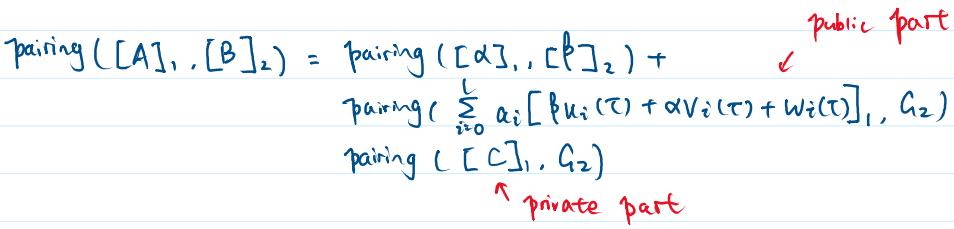
$$\text{pairing}([A]_1, [B]_2) = \text{pairing}([C]_1, G_2) + \text{pairing}([\alpha]_1, [\beta]_2)$$

Q2: What if we don't use gamma and delta?

Say we want to separate public inputs and private inputs:



Verifier will verify:

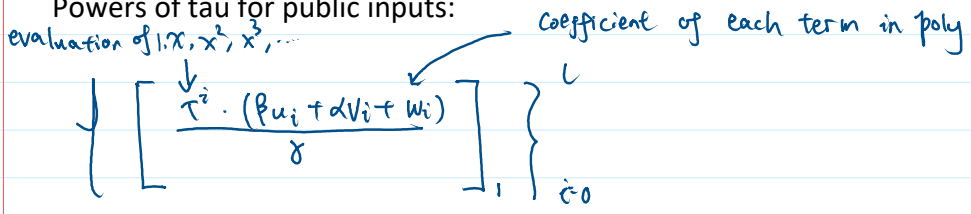


Without gamma and delta, there is no way to stop prover from using the public inputs directly when generating proof. The severity of this problem isn't as bad as the problem solved by alpha and beta, so this gamma and delta is just another layer of security.

A2: Introducing gamma and delta

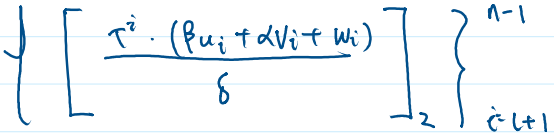
As we saw earlier, gamma and delta are introduced to stop prover from using the public inputs in the proof.

Powers of tau for public inputs:



(The only difference with alpha+beta step is division by gamma)

Powers of tau for private inputs:



Powers of tau for h(tau)t(tau)

$$\left\{\left[\frac{\tau^i \cdot t(\tau)}{\delta}\right]_1\right\}_{i=0}^{n-2}$$

Also, trusted setup needs to provide "encrypted" gamma and delta:

$$[\gamma]_2, \quad [\delta]_2$$

Verifier step updates accordingly:

$$\text{pairing}([A]_1, [B]_2) = \text{pairing}([\alpha]_1, [\beta]_2) + \text{pairing}\left(\sum_{i=0}^{l} a_i \left[\beta u_i(\tau) + \alpha v_i(\tau) + w_i(\tau)\right]_1, [\gamma]_2\right) + \text{pairing}([C]_1, [\delta]_2)$$

Cancel out $\frac{1}{\gamma}$

cancel out $\frac{1}{\delta}$

This is how gamma and delta ensures prover only follows the guidance of trusted setup and does not try to use public input directly: since prover does not know gamma and delta, if he uses public inputs directly, he can't forge a fake proof that cancels out gamma and delta terms in the pairing => pairing check will fail.

Q3: What if we don't use r and s?

The scheme isn't perfect now since attacker can brute force witness vector if he can find a certain pattern. For example, he can observe a valid proof, then brute force all possible witness vector until he finds a proof that matches the valid proof. This attack breaks the zero-knowledgeness of the scheme.

This attack is even more dangerous for scenarios with small input space, for example, voting (inputs are either 0 or 1).

A3: Introducing r and s

r and s are another set of random shifts, just like alpha and beta

$$[A]_1 = [\alpha]_1 + \sum_{i=0}^{m} a_i [u_i(\tau)]_1 + r[\gamma]_1$$

$$[B]_2 = [\beta]_2 + \sum_{i=0}^{m} a_i [v_i(\tau)]_2 + s[\delta]_2$$

$$\Rightarrow \text{new: } [B]_1 = [\beta]_1 + \sum_{i=0}^{m} a_i [v_i(\tau)]_1 + s[\delta]_1$$

$\Rightarrow$ new: $[B]_1 = [\beta]_1 + \sum_{i=0}^{m} a_i [v_i(\tau)]_1 + s[\delta]_1$

$$[C]_1 = \sum_{i=0}^{m} a_i [\beta u_i(\tau) + \alpha v_i(\tau) + w_i(\tau)]_1 + [h(\tau)t(\tau)]_1 + s[A]_1 + r[B]_1 - rs[\delta]_1$$

Prover computes $proof = ([A]_1, [B]_2, [C]_1)$

Verifier step stays the same. For simplicity, assume alpha=beta=0 and gamma=delta=1 and do the derivation. Just prove A * B = C (since alpha=beta=0).