zk-SNARKs Explainer for Developers

A zk-SNARK is a form of Zero-Knowledge Proof (ZKP), It allows someone with a secret to convince another person that they have the secret, without revealing that secret*.
Here we will see a semi-technical example of using zk-SNARKs, treating all the difficult bits as black boxes.

The Cast: Alice

This is Alice.

Alice enjoys shopping at an exclusive fashion store. The store is so exclusive that customers must know a secret passphrase to get in.

Alice knows the secret phrase is "Hello World" (which we will call **'w'**). Alice doesn't want to reveal the secret **'w'**, but is happy to publicly share the hash of this value, which we will call **'x'**. So we can write: **x = hash(w)**.

The Cast: Bob

This is Bob.

Bob has been barred from the exclusive fashion store for reasons too shocking to document, but he still works there as a store guard.

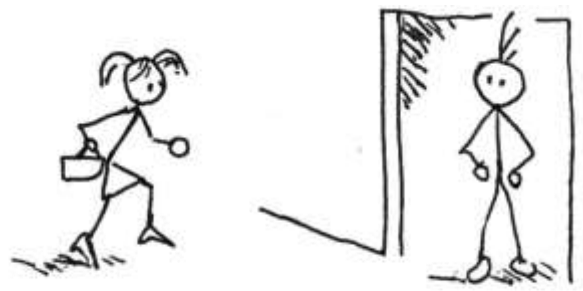
He has to let people in who know the secret phrase, but can't know the phrase himself in case he tries to get into the store again. And we don't want a repeat of last time.

Bob does know **'x'**.

The Scenario

Alice would like to visit the store and Bob is on duty. In other scenarios Alice could simply say the passphrase, but here she cannot.

zk-SNARKs to the rescue! Instead of revealing the secret **'w'**, Alice will reveal a proof that she knows **'w'**.



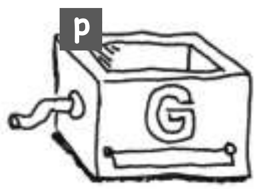
The Setup

Let's do some setup. First we need to create a check program called **'*C*'** that we can use to convince ourselves (and others) that a person knows the secret phrase **'w'** if we are given the public value **'x'**. In pseudo-code we can use:

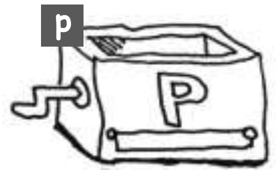
```
public bool C(x, w) {
    return (hash(w) == x);
}
```

If the correct secret **'w'** is given, then the above will return true. We also setup three algorithms which we will treat as black boxes.

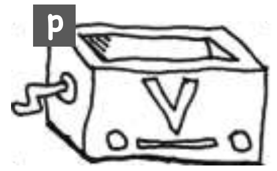
We use the icon p to identify public information, and ⃠ to identify secret information.



Generator Algorithm



Prover Algorithm



Verifier Algorithm

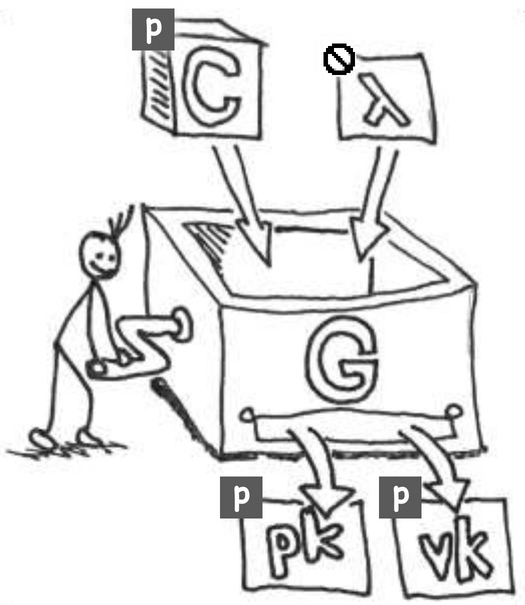
* Strictly speaking, the person with the secret convinces the other person that the secret conforms to some rule (the program *C* you see later).

The Verifier

Bob is the verifier.

He passes the program '**C**' and a value '**λ**' lamba into the **Generator** algorithm, to produce a proving key ('**pk**') and verification key ('**vk**').

The value '**λ**' lamba is a secret value known only to Bob**. The '**pk**' and '**vk**' are public.

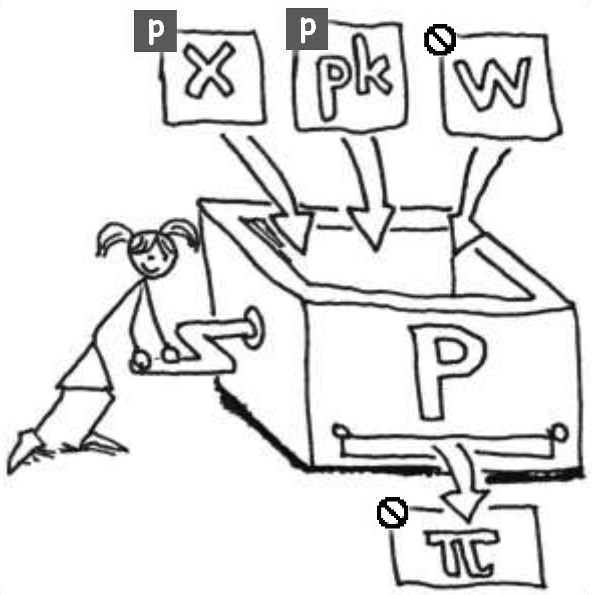


The Prover

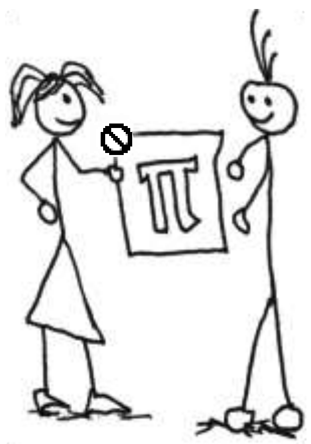
Alice is the prover.

She passes '**pk**', '**x**' (the hash of '**w**') and her secret '**w**' into the **Prover** algorithm, to produce a proof '**Π**'.

The proof '**Π**' should not be public, Alice should reveal it only to Bob.



Alice passes the proof to Bob.

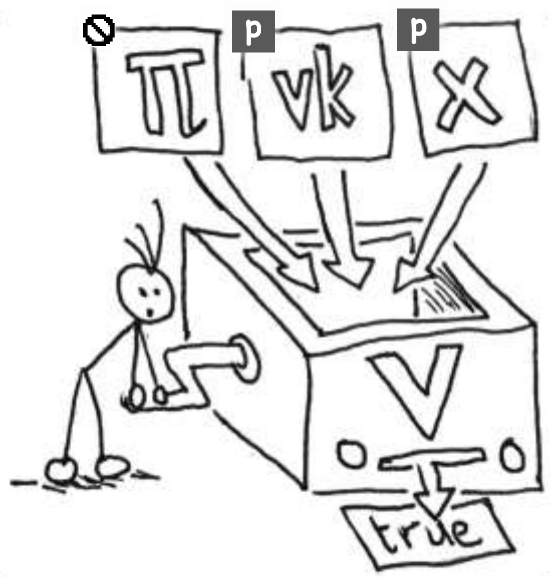


The Verification

Bob can now verify if Alice should be allowed to enter the store.

He passes the proof '**Π**' he got from Alice, '**vk**' and '**x**' into the **Verifier** algorithm.

The verifier algorithm produces the value **true** meaning Alice can enter the store.



Store access granted!

That's a Wrap

Note that Bob cannot use the proof '**Π**' to convince his fellow store guard Chloe that he knows the secret phrase.

This is because when Chloe acts as a verifier, she would generate her own '**pk**', '**vk**' pair using her own secret '**λ**' lambda.

** or others could create fake proofs that make it appear that they know secret '**w**' when they don't. Think of '**λ**' lamba as 'toxic waste'.