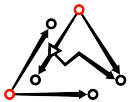


Least Authority
PRIVACY MATTERS

Tutorial: Attacks on Zero-Knowledge Proof Systems

ZKProof5 | November 16, 2022 | Tel Aviv, Israel

Presented by: Anna Kaplan

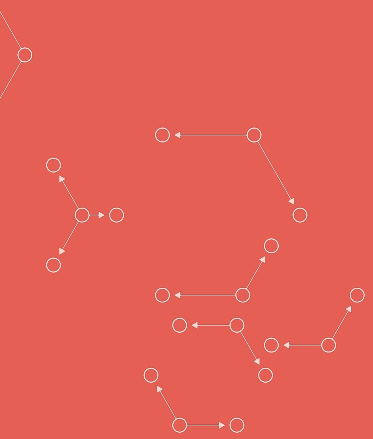
Outline

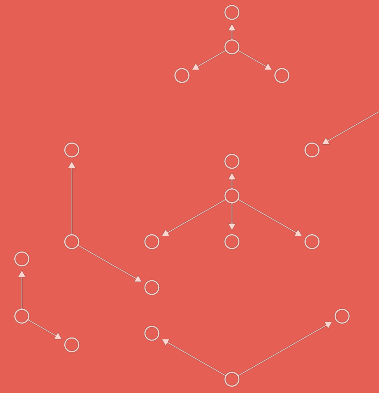
Example 1: Un-Trusted Setup

Example 2: Documentation Gone Wrong

Example 3: Primitive Primitives

Other Examples & What's Next?





01

Example 1: Un-Trusted Setup

∃ zk-SNARKs that need a trusted setup

Preaching to the choir!

Trusted setup: In a setup phase for the zk-SNARK, a common reference string and a simulation trapdoor are generated. The simulation trapdoor is also called toxic waste.

Either:

Perform a MPC or

use a zero-knowledge proof system which doesn't need a trusted setup!

Don't use a system which needs a trusted setup without one!

Silly analogy

Map: Google Maps Screenshot
Plan: https://www.flickr.com/photos/adavey/49064230968
Explorer: https://www.ace.co.il/4419897 (Only 279 NIS!)

Learnings

DOs & DON'Ts

👍Perform a trusted setup when needed

👍Understand the boundaries and mathematical models of the trusted setup situation you use

👍Review your implementation for a trusted setup

👍Keep up to date with the research community

👎Do not not document it

02

Example 2: Documentation Gone Wrong

Document what your statement in a mathematical way!

For zk-SNARKs, we say: "zk-SNARKs are succinct non-interactive arguments of knowledge, where the knowledge-proof attests to the correctness of a statement."

But what is a **statement**?

My zk-SNARK implementation proves that I have knowledge over inputs x, y, and b such that if b is TRUE, then I can output the sum of x and y, and if b is FALSE, then I can output the multiplication of x and y.

{An example statement}

Example: Please write this example in pseudo code

My zk-SNARK implementation proves that I have knowledge over inputs x, y, and b such that if b is TRUE, then I can output the sum of x and y, and if b is FALSE, then I can output the multiplication of x and y.

{An example statement}

Example: Please write this example in pseudo code

My zk-SNARK implementation proves that I have knowledge over inputs x, y, and b such that if b is TRUE, then I can output the sum of x and y, and if b is FALSE, then I can output the multiplication of x and y.

{An example statement}

```
function EXAMPLE-STATEMENT-V0(x, y, b)
    z ← 0
    if b = 1 then
        z ← x + y
    else
        z ← x · y
    end if
    return z
end function
```

Example: Please write this example in pseudo code

My zk-SNARK implementation proves that I have knowledge over inputs x, y, and b such that if b is TRUE, then I can output the sum of x and y, and if b is FALSE, then I can output the multiplication of x and y.

{An example statement}

```
1 fn example_statement(x: F, y: F, b: F) -> F
2 {   let mut z: F = 0;
3     if b == 1 {
4         z = add(x,y);
5     }
6     else {
7         z = mul(x,y);
8     }
9     return z;
10 }
```

function EXAMPLE-STATEMENT-V0(x, y, b)
  $z \leftarrow 0$
  if $b = 1$ **then**
    $z \leftarrow x + y$
  else
    $z \leftarrow x \cdot y$
  end if
  return z
end function

Example: Please write this example in pseudo code

My zk-SNARK implementation proves that I have knowledge over inputs x, y, and b such that if b is TRUE, then I can output the sum of x and y, and if b is FALSE, then I can output the multiplication of x and y.

{An example statement}

function EXAMPLE-STATEMENT-V0(x, y, b)
    $z \leftarrow 0$
    if $b = 1$ **then**
        $z \leftarrow x + y$
    else
        $z \leftarrow x \cdot y$
    end if
    return z
end function

```
fn example_statement(x: F, y: F, b: F) -> F
{   let mut z: F = 0;
    if b == 1 {
        z = add(x,y);
    }
    else {
        z = mul(x,y);
    }
    return z;
}
```

```
fn example_statement(x: F, y: F, b: bool) -> F
{   let mut z: F = 0;
    if b {
        z = add(x,y);
    }
    else {
        z = mul(x,y);
    }
    return z;
}
```

Example: Please draw an arithmetic circuit from this statement

```
function EXAMPLE-STATEMENT-V0(x, y, b)
    z ← 0
    if b = 1 then
        z ← x + y
    else
        z ← x · y
    end if
    return z
end function
```

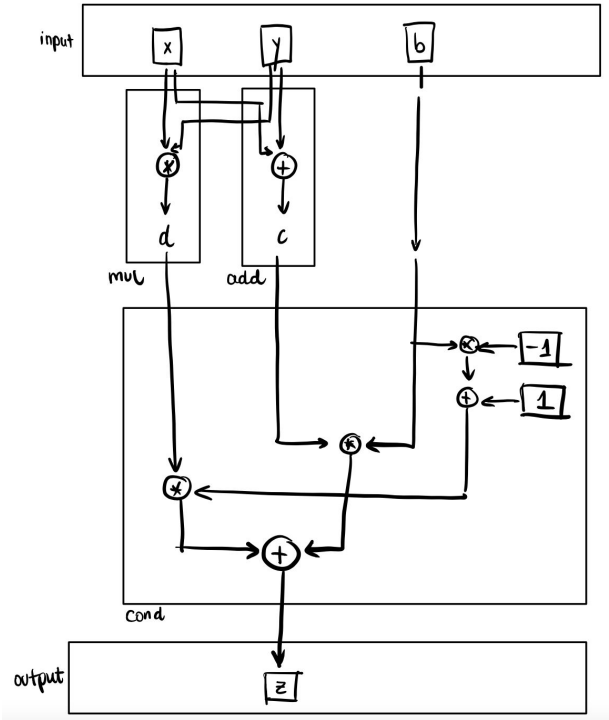
Reminder 1:
An arithmetic circuit consists of addition and multiplication gates over a finite field F.

Reminder 2:
If/Else can be represented, for b a Boolean:

outcome = b * if-true-value + (1-b) * else-value

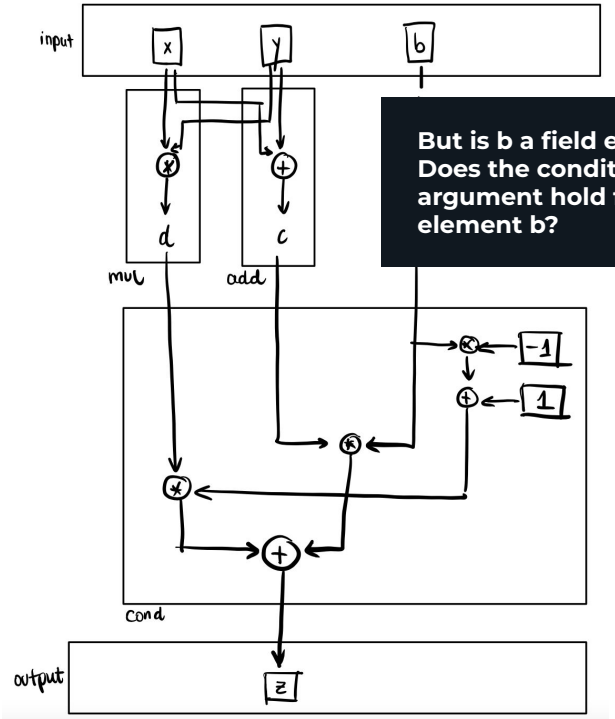
Example: Please draw an arithmetic circuit from this statement

```
function EXAMPLE-STATEMENT-V0(x, y, b)
    z ← 0
    if b = 1 then
        z ← x + y
    else
        z ← x · y
    end if
    return z
end function
```



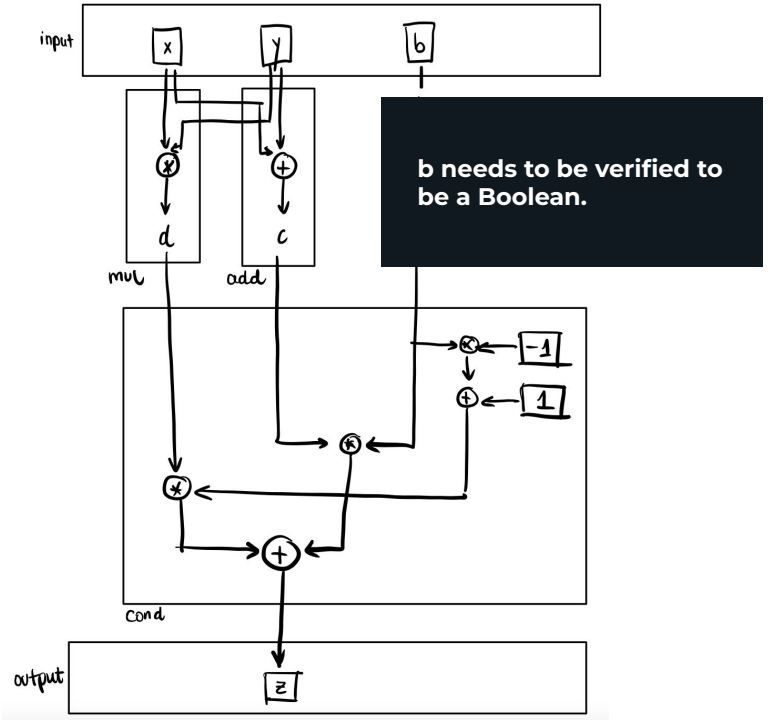
Example: Please draw an arithmetic circuit from this statement

```
function EXAMPLE-STATEMENT-V0(x, y, b)
    z ← 0
    if b = 1 then
        z ← x + y
    else
        z ← x · y
    end if
    return z
end function
```



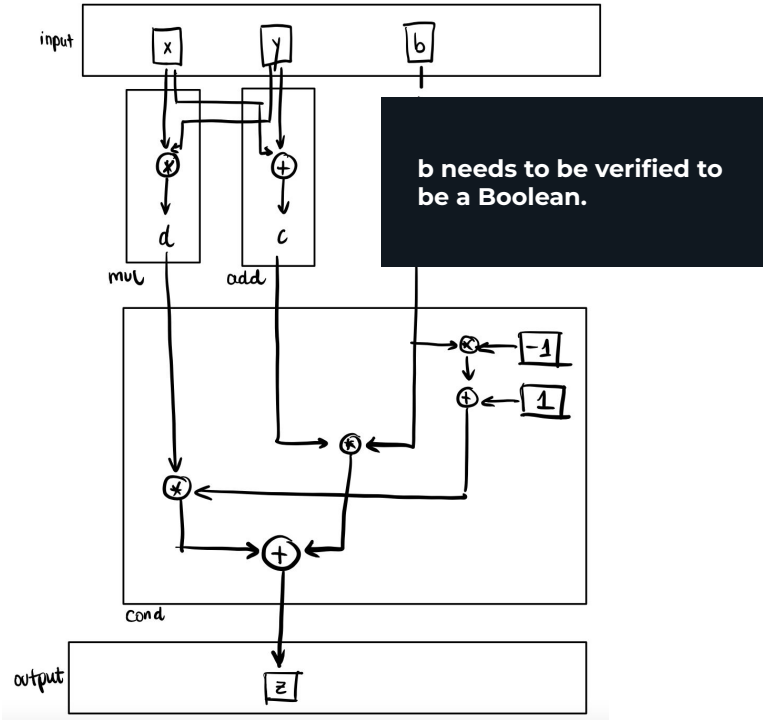
Example: Please draw an arithmetic circuit from this statement

```
function EXAMPLE-STATEMENT-V0(x, y, b)
    z ← 0
    if b = 1 then
        z ← x + y
    else
        z ← x · y
    end if
    return z
end function
```



Example: Please draw an arithmetic circuit from this statement

```
function EXAMPLE-STATEMENT-V0(x, y, b)
    z ← 0
    if b = 1 then
        z ← x + y
    else
        z ← x · y
    end if
    return z
end function
```



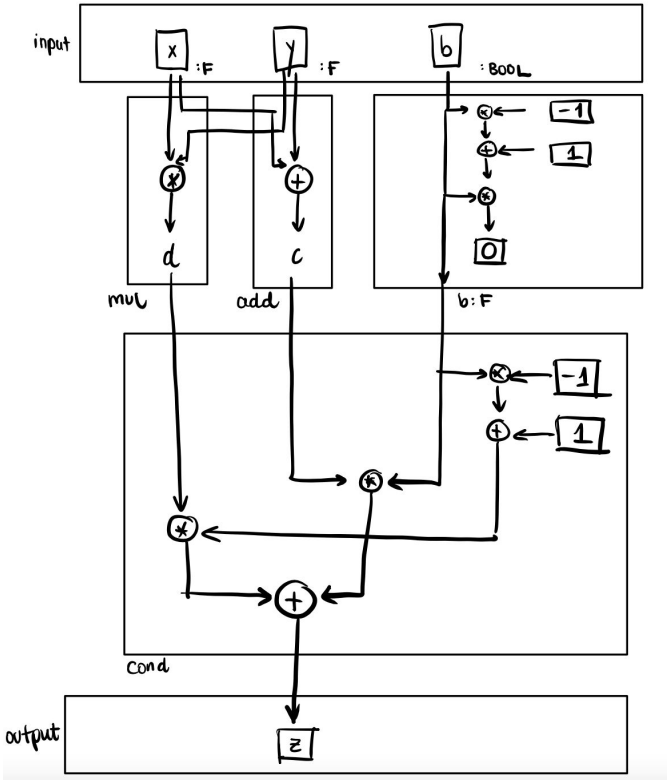
Reminder 3:

For a Boolean b, this holds:

(1-b) * b = 0

Example: Please draw an arithmetic circuit from this statement

```
function EXAMPLE-STATEMENT-V0(x, y, b)
    z ← 0
    if b = 1 then
        z ← x + y
    else
        z ← x · y
    end if
    return z
end function
```



Reminder 3:

For a Boolean b, this holds:

(1-b) * b = 0

Example: Please write this example in pseudo code

My zk-SNARK implementation proves that I have knowledge over inputs x, y, and b such that if b is TRUE, then I can output the sum of x and y, and if b is FALSE, then I can output the multiplication of x and y.

{An example statement}

```
function EXAMPLE-STATEMENT-V0(x, y, b)
    z ← 0
    if b = 1 then
        z ← x + y
    else
        z ← x · y
    end if
    return z
end function
```

No types specified!
Constants & field not specified!
Why are we doing this?!

Learnings

DOs & DON'Ts

👍 Cite and explain your references

👍 Include a high-level explanation of your system

👍 Explain your notation

👍 Use graphics where you can

👍 Include technical concerns you had and their resolution

👍 Include an information flow

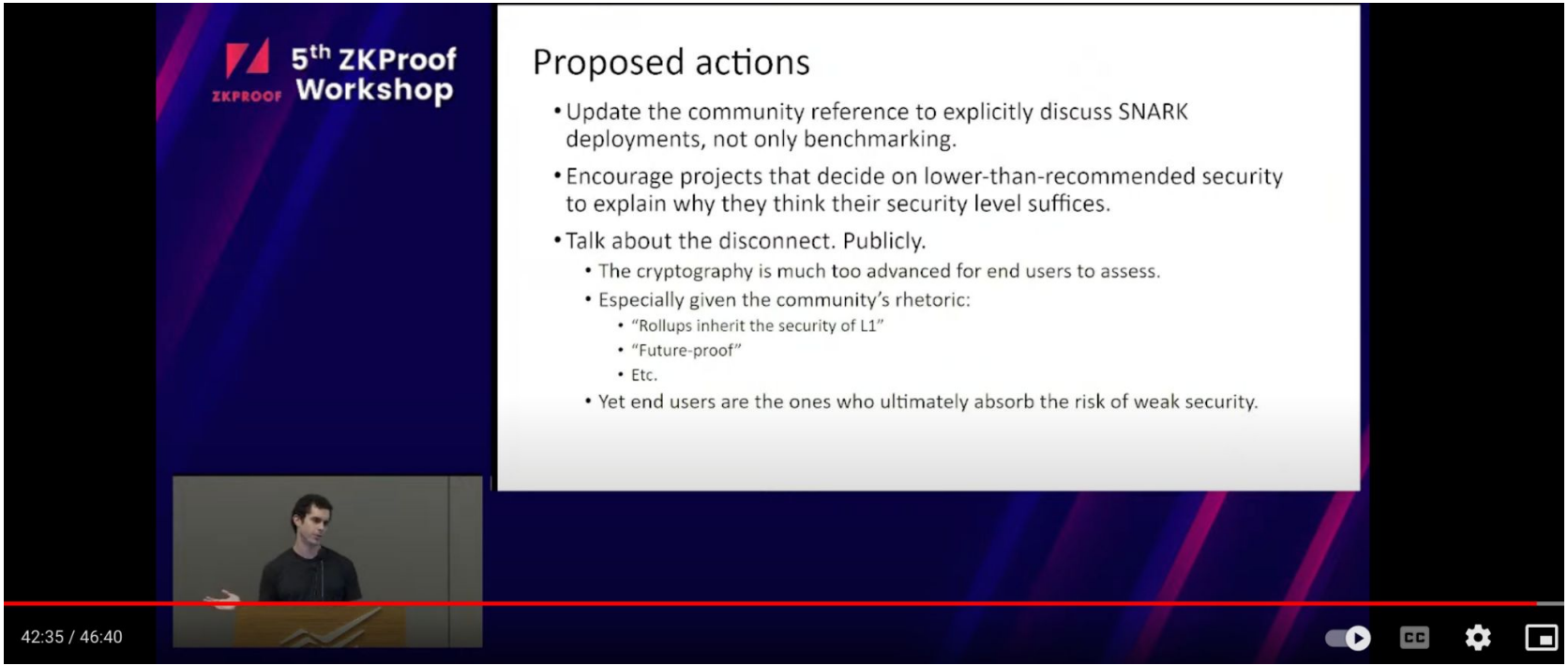
👍 Include descriptions and justification of used parameters

👍 Perform or reference an analysis of the system (e.g. a security proof) with mentioning modelling choices for the performed analysis

👎 Only document the updates – especially if you don't adhere to the original notation

👎 Only provide parameters in the specification without a justification, discussion or comment

Learnings, from yesterday



03

Example 3: Primitive Primitives

Zero-knowledge proof systems use intrinsic cryptographic primitives most of the time!

When I say cryptographic primitive, what comes to your mind?

Zero-knowledge proof systems use intrinsic cryptographic primitives most of the time!

When I say cryptographic primitive, what comes to your mind?

E.g.: Elliptic curves, hash functions, encryption schemes, signature schemes, ...

Challenge: Choosing an elliptic curve

BLS6-6

BN254

BLS12-381

Challenge: Choosing a hash function

MiMC

Blake2

Poseidon

SHA256

Learnings

DOs & DON'Ts

👍 Know your use case and its boundaries

👍 Compare primitives to each other

👍 Engage in research regarding the primitives you're using

👍 Keep up to date with the research community

👍 Understand the boundaries and mathematical models of the used primitives

👎 Don't tweak cryptographic primitives

04

What's Next?

What's next?

Other attacks on zero-knowledge proof systems:

- Forging proofs by leaked keys
- Malleability of zero-knowledge proof systems
- Replay attacks
- etc.

Have you seen any other attacks in the wild?

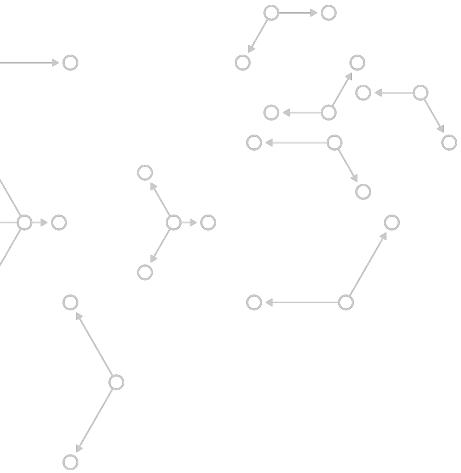
What's Next?

Other attacks on zero-knowledge proof systems:

- Forging proofs by leaked keys
- Malleability of zero-knowledge proof systems
- Replay attacks
- etc.

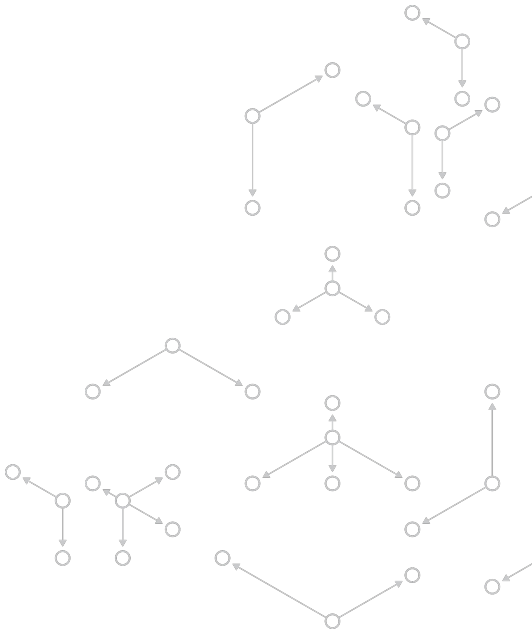
Have you seen any other attacks in the wild?

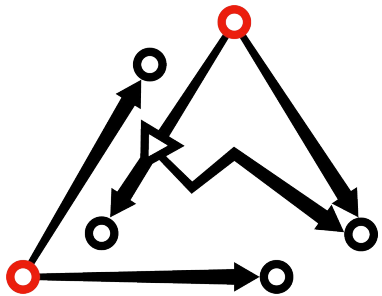
Join our effort to collect these and sum up development strategies at Least Authority!



Anna Kaplan
anna@leastauthority.com
@kaplannie

Thanks for coming and keep in touch!
@leastauthority





01.1
Backup Slides

Example: Please draw an algebraic circuit from this statement

```
statement CONDITIONAL_OP {F:F_p} {
  fn main(x : F, y : F, b : BOOL) -> F {
    let z : F
    z <== if b then {
      ADD(x,y)
    } else {
      MUL(x,y)
    } ;
    return z ;
  }
}
```

Make sure to make b a BOOLEAN!

Example: Please draw an algebraic circuit from this statement

```
statement CONDITIONAL_OP {F:F_p} {
  fn main(x : F, y : F, b : BOOL) -> F {
    let z : F
    z <== if b then {
      ADD(x,y)
    } else {
      MUL(x,y)
    } ;
    return z ;
  }
}
```

Tip:
If/Else can be represented, for b a Boolean:

outcome = b * if-true-value + (1-b) * else-value

But also:
For a Boolean b, this holds:

(1-b) * b = 0