C: There's a SNARK for That

Alessandro Chiesa
Daniel Genkin
Eli Ben-Sasson
Eran Tromer
Madars Virza
& Co.

Problem: integrity on untrusted platform

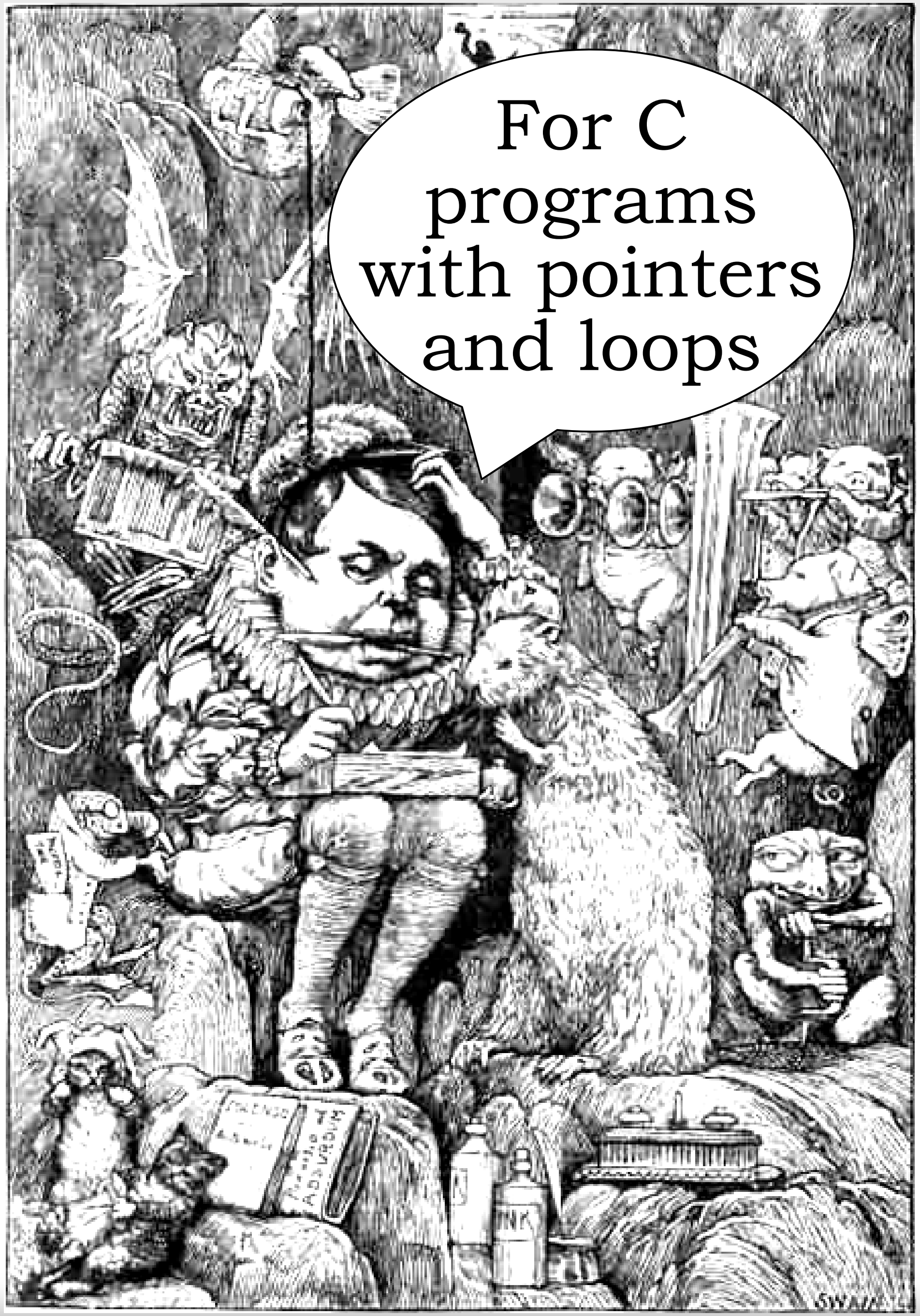
- Faults
- Someone else’s cloud
- Platform trojans
- Blue pill
- OS bugs
- Untrusted data origins
- Crypto protocols

Solution: zk-SNARKs

zero knowledge
Succint
Noninteractive
Argument of
Knowledge

AKA

- Non-interactive CS proofs of knowledge
- Succint NIZK



RC4.c

```c
#include <tinyram.h>
#define LEN 16

int main() {
    int state[256], stream[LEN];
    int i, j, t, k;

    for (i=0; i < 256; ++i)  state[i] = i;
    /* KSA: mix in key */
    k = 0; j = 0;
    for (i=0; i < 256; ++i) {
        t = state[i];
        keybyte = read_aux_input_tape();
        j = (j + t + keybyte) & 0xFF;
        state[i] = state[j]; state[j] = t;
    }

    /* PRGA: produce stream */
    i=0; j=0;
    for (k=0; k < LEN; k++)  {
        i = (i + 1) & 0xFF;
        t = state[i];
        j = (j + t) & 0xFF;
        state[i] = state[j];
        state[j] = t;
        stream[k] = state[(state[i] + state[j]) & 0xFF];
    }

    /* compare with the claim */
    for (i=0; i < LEN; i++)
        if (stream[i] != read_primary_input_tape()) {
            return 1;
    return 0;
}
```

SOUTH POLE EQUINOX **EAST** ZENITH LONGITUDE

EQUATOR

NORTH

LATITUDE

NADIR NORTH POLE **WEST** MERIDIAN TORRID ZONE

SOUTH

Compass-Points. N, E, S, W.

Scale

C program

↓

Compiler — based on GCC

↓

TinyRAM program

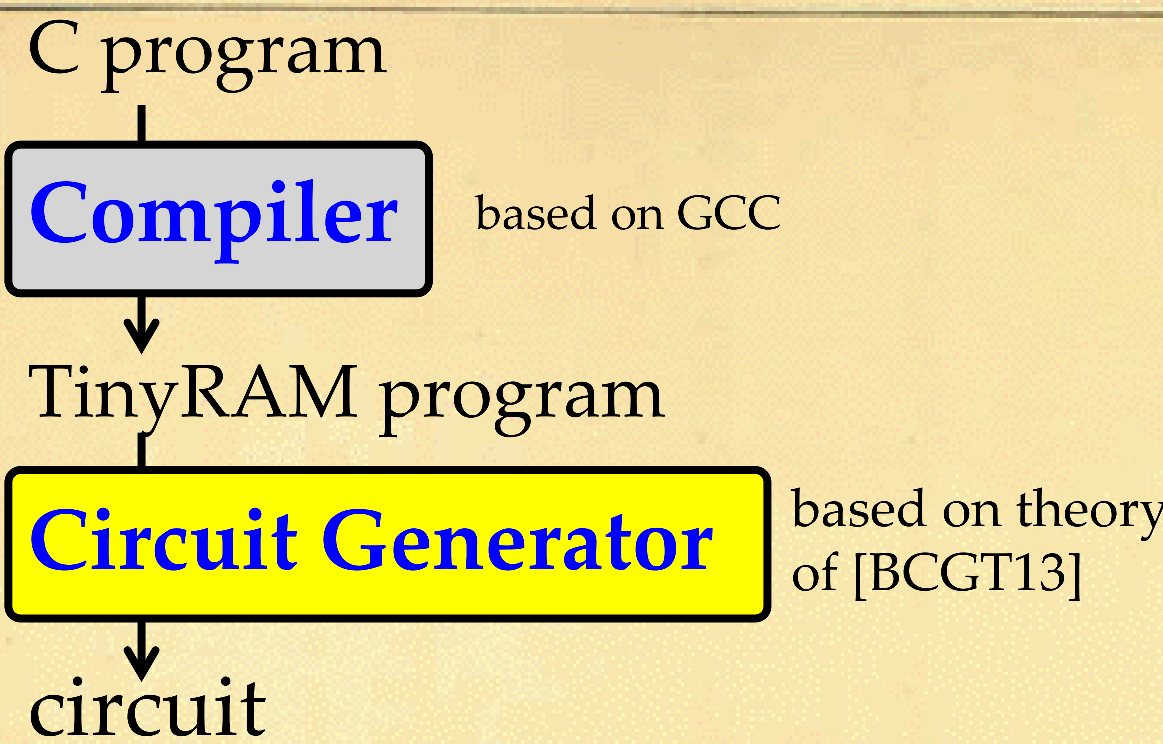
TinyRAM architecture for fast verification

instruction mnemonic	operands	effects	flag	notes
and	ri rj A	compute bitwise AND of $[rj]$ and $[A]$ and store result in ri	result is 0^W	
or	ri rj A	compute bitwise OR of $[rj]$ and $[A]$ and store result in ri	result is 0^W	
xor	ri rj A	compute bitwise XOR of $[rj]$ and $[A]$ and store result in ri	result is 0^W	
not	ri A	compute bitwise NOT of $[A]$ and store result in ri	result is 0^W	
add	ri rj A	compute $[rj]_u + [A]_u$ and store result in ri	overflow	
sub	ri rj A	compute $[rj]_u - [A]_u$ and store result in ri	borrow	
mull	ri rj A	compute $[rj]_u \times [A]_u$ and store least significant bits of result in ri	overflow	
umulh	ri rj A	compute $[rj]_u \times [A]_u$ and store most significant bits of result in ri	overflow	
smulh	ri rj A	compute $[rj]_s \times [A]_s$ and store most significant bits of result in ri	over/underflow	
udiv	ri rj A	compute quotient of $[rj]_u/[A]_u$ and store result in ri	$[A]_u = 0$	
umod	ri rj A	compute r[illegible]	[illegible]	
shl	ri rj A	shift $[rj]$ b		
shr	ri rj A	shift $[rj]$ b		
cmpe	ri A	none ("con		
cmpa	ri A	none ("con		
cmpae	ri A	none ("con		
cmpg	ri A	none ("con		
cmpge	ri A	none ("con		
mov	ri A	store $[A]$ in		
cmov	ri A	if flag = 1,		
jmp	A	set pc to [		
cjmp	A	if flag = 1,		
cnjmp	A	if flag = 0,		
store	A ri	store $[ri]$ at memory address $[A]_u$		
load	ri A	store the content of memory address $[A]_u$ into ri		
read	ri A	if the $[A]_u$-th tape has remaining words then consume the next word, store it in ri, and set flag = 0; otherwise store 0^W in ri and set flag = 1	$\longleftarrow$	(1)
answer	A	stall or halt (and the return value is $[A]_u$)		(2)

(1) All but the first two tapes are empty: if $[A]_u \notin \{0,1\}$ then store 0^W in ri and set flag = 1.
(2) answer causes a stall (i.e., not increment pc) or a halt (i.e., the computation stops); the choice between the two is undefined.

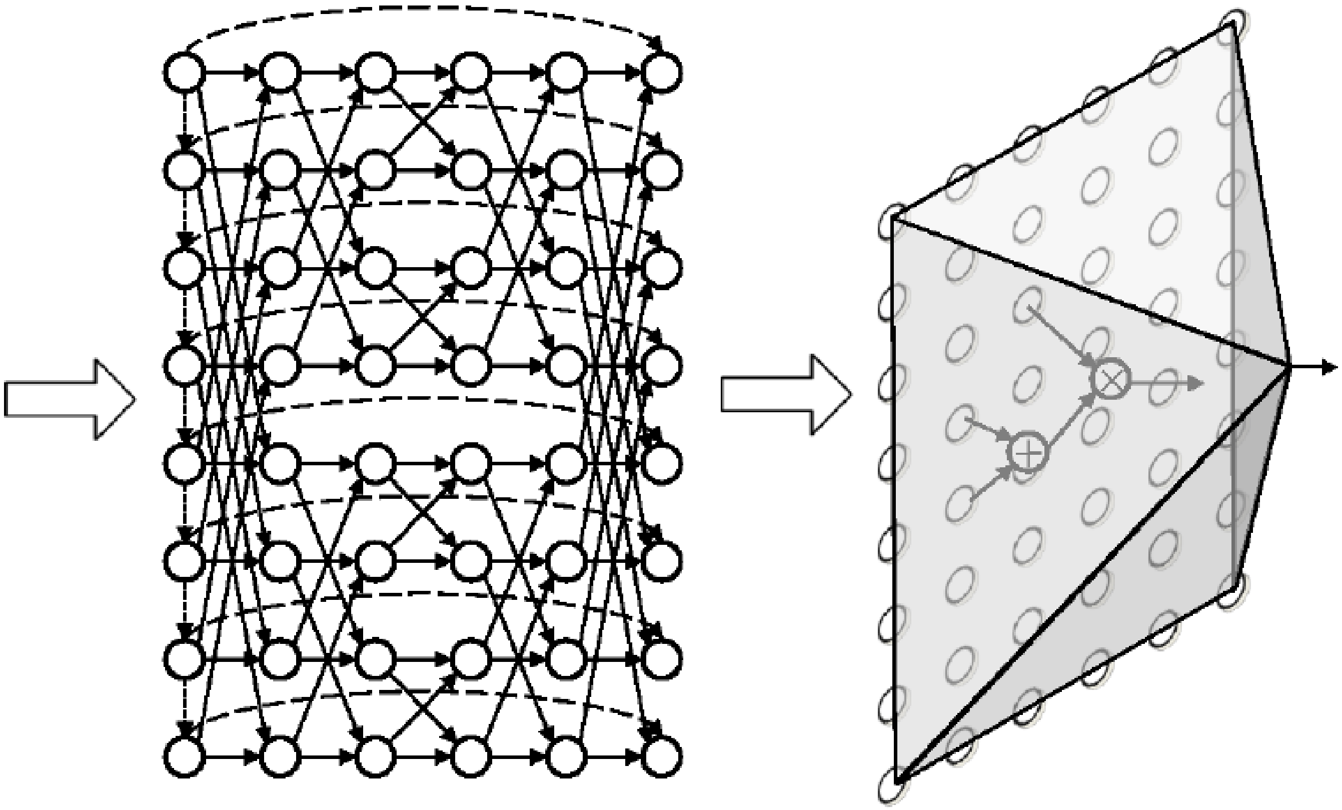
$+, -, \times, div, mod, \oplus$, shifts
Arithmetic comparison,
Branches
Memory load/store
Input tapes read

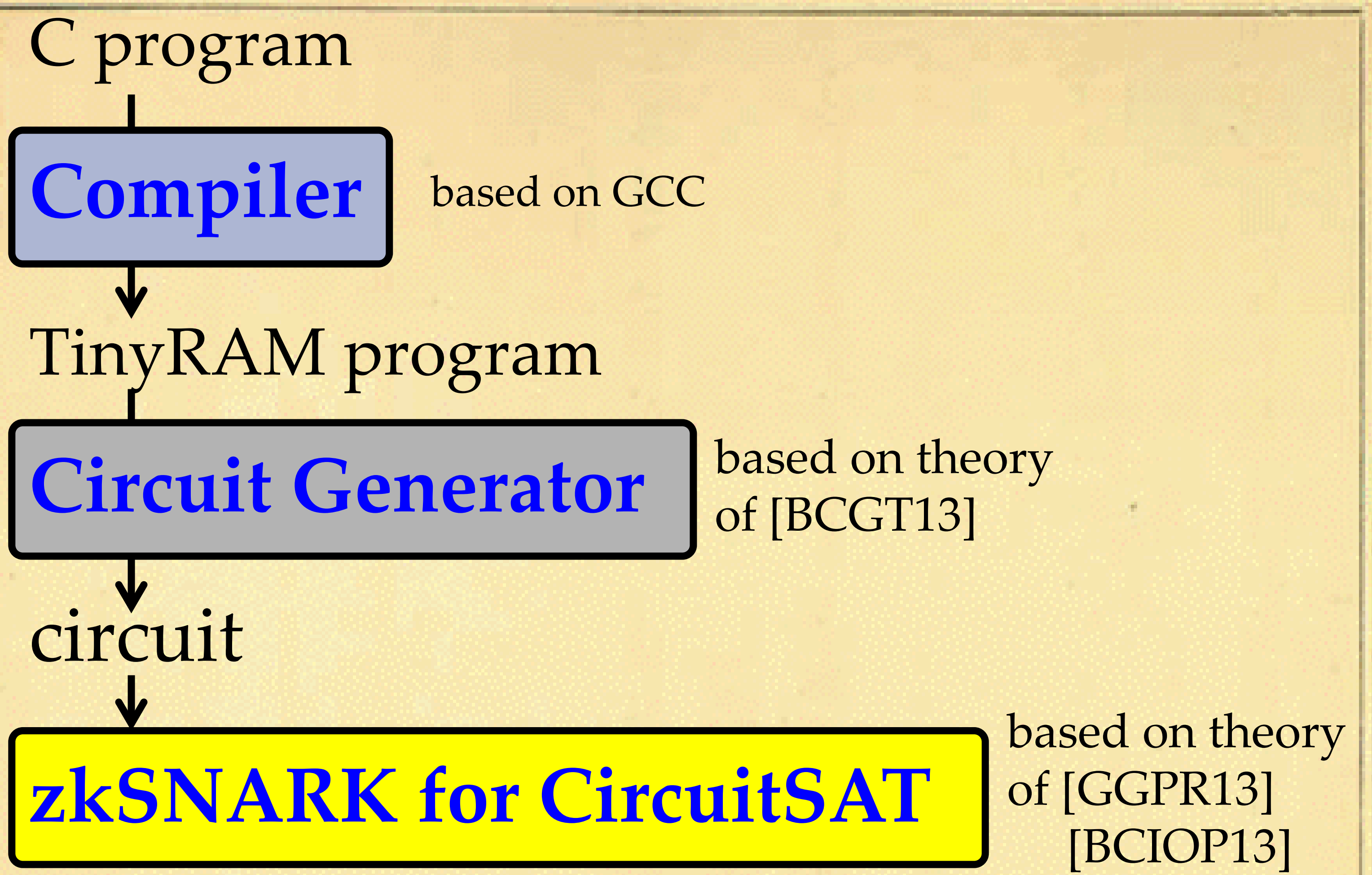
Spec: **http://scipr-lab.org/tinyram**


C program
Compiler
based on GCC
TinyRAM program
Circuit Generator
based on theory of [BCGT13]
circuit

Converting TinyRAM verification to circuit satisfiability

```
_sumarray:
     store r0   r0
     mov   r1   0000
     mov   r2   1000
     mov   r3   2000
     mov   r4   0
     mov   r5   100
_loop:
     cmpe  r4   r5
     cjmp  _end
     load  r6   r1
     pload r6   r1
     load  r7   r2
     pload r7   r2
     add   r8   r7   r5
     store r3   r8
     add   r1   r1   1
     add   r2   r2   1
     add   r3   r3   1
     add   r4   r4   1
     jmp   _sum
_end:
```




C program
Compiler
based on GCC
TinyRAM program
Circuit Generator
based on theory of [BCGT13]
circuit
zkSNARK for CircuitSAT
based on theory of [GGPR13] [BCIOP13]

"I know an RC4 key producing 26 41 5B C4 4C EC ED 6C 89 99 68 E1 82 04 DE"

322-byte proof

"The execution of arbitrary C programs can be verified in a few milliseconds and 322 bytes"

http://scipr-lab.org

What Would you Like to Prove Today?