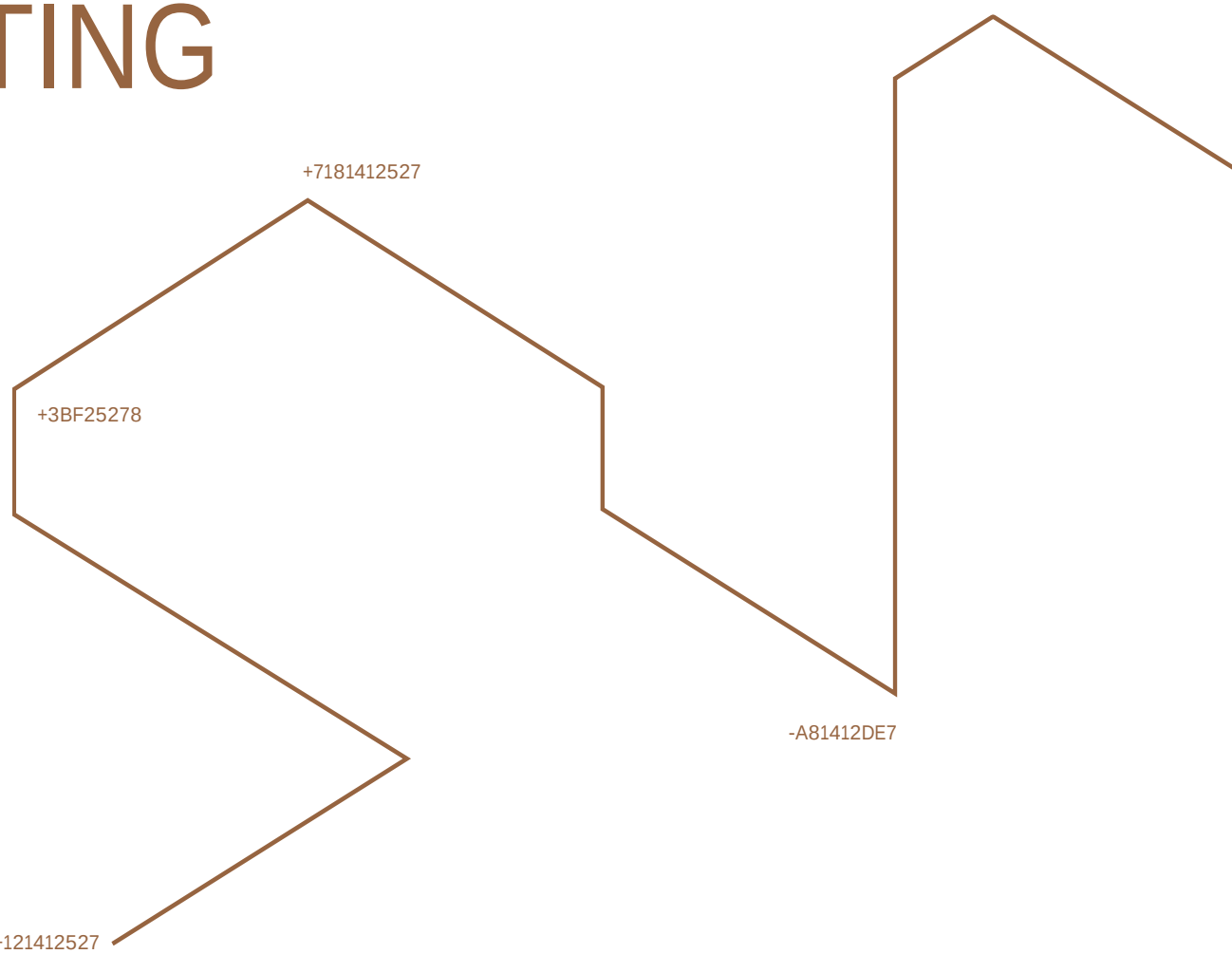
ACCU 2022

INTRODUCTION TO VERIFIABLE COMPUTING

AHTO TRUU

guardtime

VERIFIABLE COMPUTING
AN INTRODUCTION

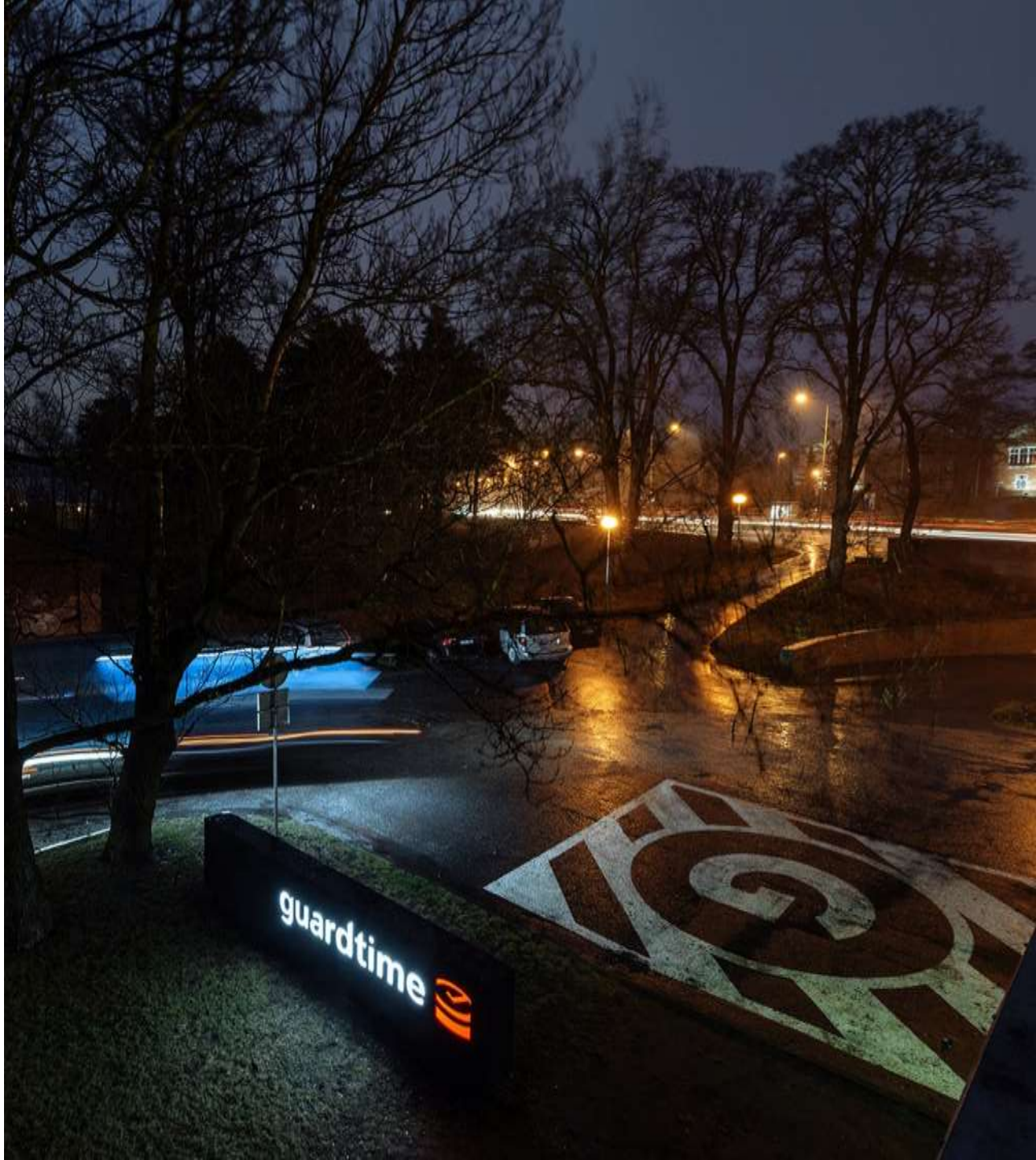


AHTO TRUU
SOFTWARE ARCHITECT, GUARDTIME

ACCU CONFERENCE, 09-APR-2022

ABOUT GUARDTIME

+ Systems engineering company focusing on data security solutions
+ Founded in 2007 in Tallinn, Estonia
+ Global HQ in Lausanne, Switzerland
+ Offices in US, EU and China
+ 150 employees
+ 80% engineers and researchers
+ https://guardtime.com/



AGENDA

- Motivation
- Cryptographic tools
- Arithmetic circuits and programs
- Verifiable computations

guardtime

1/ MOTIVATION

VERIFIABLE COMPUTING

- A client outsources some work to a server
- The server returns the results
  - and a proof that the results are correct

- Has also been called
  - Certified computation
  - Delegation of computation
  - Computational integrity

AUTHENTICATED DATA STRUCTURES

- The server maintains a database for the client
- For every update request
  - proof that the update was applied
  - proof that nothing else was changed
- For every query
  - proof that the results match the current state of the database

GENERAL COMPUTATIONS

- A client outsourcing to a cloud provider
- IoT devices supported by external servers
- Smart contracts executed on a blockchain

- Useful when verification is cheaper than execution
- For some computations this comes naturally
- For the rest, extra work on the server side

INTEGRITY VS CONFIDENTIALITY

- What parties are allowed to see what data?

- Verifiable electronic voting
- Privacy-preserving meter readings
- Private transactions on a blockchain

guardtime

2/ CRYPTOGRAPHIC TOOLS

DISCRETE LOGARITHM PROBLEM

- Given m, g and y, find x such that $g^x \bmod m = y$
- Trivial to solve without the modulus
- Can be infeasibly difficult in the “modular” form
- Solution efficiently verifiable using the “square and multiply” method
  - to compute g^{13}
  - observe that $13 = 1101_2$
  - compute g^2, g^4, g^8, then $g^{13} = g \cdot g^4 \cdot g^8$

CRYPTOGRAPHIC COMMITMENTS

- Fix an input x, compute and publish commitment $y = C(x)$
- Later "open" x and prove relationship to y
- Binding and hiding properties

- Naïve: compute $y = g^x \bmod m$
- Pedersen: pick random r, compute $y = g^x \cdot h^r \bmod m$

- Observe that in both cases $C(x_1 + x_2) = C(x_1) \cdot C(x_2)$

ZERO-KNOWLEDGE PROOFS

- Prove that you know something
- But without revealing that something

- Digital signatures
  - creation of a signature only possible with private key
  - signature verification proves the signer knows the key
  - the key itself not revealed in the process

- Recall also $C(x_1 + x_2) = C(x_1) \cdot C(x_2)$

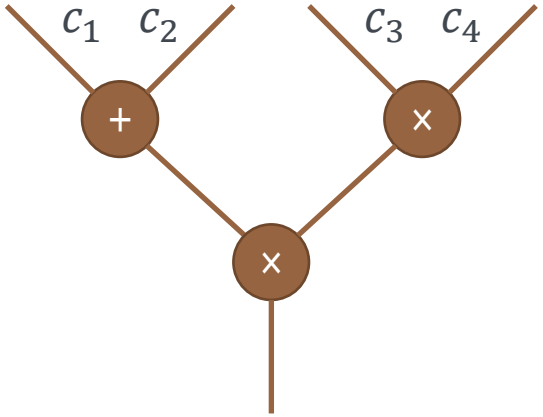
BILINEAR MAPPINGS

- Consider functions $G_1 \times G_1 \rightarrow G_2$
- Function e is a bilinear map if $e(a \cdot x, b \cdot y) = a \cdot b \cdot e(x, y)$
- If G_2 is multiplicative, the rule becomes $e(a \cdot x, b \cdot y) = e(x, y)^{a \cdot b}$

guardtime

3/ ARITHMETIC CIRCUITS AND PROGRAMS

ARITHMETIC CIRCUITS

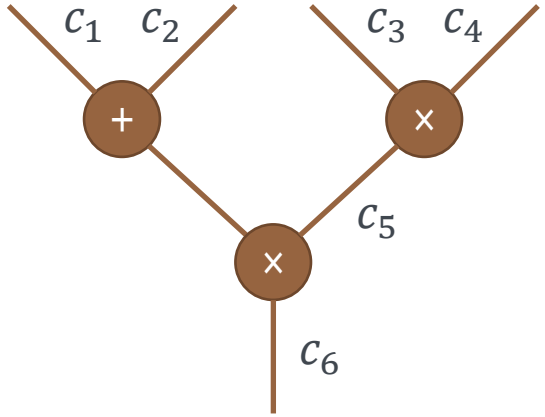


+ Representation of computations
+ Wire values are numbers
+ Gates are additions and multiplications
+ $(c_1 + c_2) \cdot (c_3 \cdot c_4)$

+ No loops: unrolled by the compiler
+ No conditionals: `a ? b : c` replaced with $a \cdot b + (1 - a) \cdot c$
+ Polynomial-sized circuits equivalent to polynomial-time Turing machines

QUADRATIC ARITHMETIC PROGRAMS (1)

- Encoding of arithmetic circuits as sets of polynomials



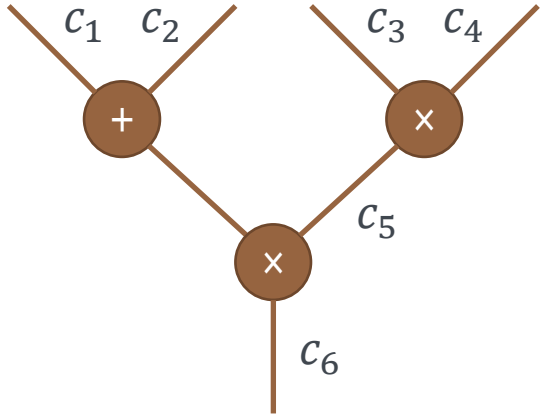
- Add labels c_5, c_6
- Pick arbitrary r_5, r_6
- Pick polynomials $v_k(x)$, $w_k(x)$, $y_k(x)$ satisfying the table on the right

	(r_5, r_6)		(r_5, r_6)		(r_5, r_6)
$v_1(r_i)$	(0,1)	$w_1(r_i)$	(0,0)	$y_1(r_i)$	(0,0)
$v_2(r_i)$	(0,1)	$w_2(r_i)$	(0,0)	$y_2(r_i)$	(0,0)
$v_3(r_i)$	(1,0)	$w(r_i)$	(0,0)	$y_3(r_i)$	(0,0)
$v_4(r_i)$	(0,0)	$w_4(r_i)$	(1,0)	$y_4(r_i)$	(0,0)
$v_5(r_i)$	(0,0)	$w_5(r_i)$	(0,1)	$y_5(r_i)$	(0,1)
$v_6(r_i)$	(0,0)	$w_6(r_i)$	(0,0)	$y_6(r_i)$	(1,0)

QUADRATIC ARITHMETIC PROGRAMS (2)

- Define
- $p(x) = (\sum c_k \cdot v_k(x)) \cdot (\sum c_k \cdot w_k(x)) - (\sum c_k \cdot y_k(x))$
- $t(x) = (x - r_5) \cdot (x - r_6)$

- Require $t(x)$ divides $p(x)$



	(r_5, r_6)		(r_5, r_6)		(r_5, r_6)
$v_1(r_i)$	(0,1)	$w_1(r_i)$	(0,0)	$y_1(r_i)$	(0,0)
$v_2(r_i)$	(0,1)	$w_2(r_i)$	(0,0)	$y_2(r_i)$	(0,0)
$v_3(r_i)$	(1,0)	$w(r_i)$	(0,0)	$y_3(r_i)$	(0,0)
$v_4(r_i)$	(0,0)	$w_4(r_i)$	(1,0)	$y_4(r_i)$	(0,0)
$v_5(r_i)$	(0,0)	$w_5(r_i)$	(0,1)	$y_5(r_i)$	(0,1)
$v_6(r_i)$	(0,0)	$w_6(r_i)$	(0,0)	$y_6(r_i)$	(1,0)

guardtime

4/ VERIFIABLE COMPUTATIONS

FORMAL DEFINITION

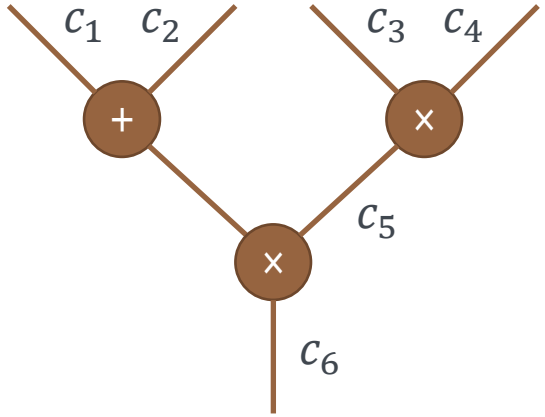
- $(EK_F, VK_F) \leftarrow \text{KeyGen}\left(F, 1^{\lambda}\right)$
- $\left(y, \pi_y\right) \leftarrow \text{Compute}(EK_F, u)$
- $\{0,1\} \leftarrow \text{Verify}(VK_F, u, y, \pi_y)$

PINOCCHIO

- Joint work of MS Research and IBM Research
  - takes a function implemented in subset of C
  - converts it into QAP representation
  - derives the key generation, computer/prover and verifier code
- Milestone result: first time ever verifier was faster than native execution

PINOCCHIO: KEY GENERATION

- $(EK_F, VK_F) \leftarrow \text{KeyGen}(F, 1^\lambda)$
- F to circuit to QAP: $t(x), V, W, Y; I_{IO}, I_{mid}, e, g$
- Pick $r_v, r_w, s, \alpha_v, \alpha_w, \alpha_y, \beta, \gamma$
- $r_y = r_v \cdot r_w, g_v = g^{r_v}, g_w = g^{r_w}, g_y = g^{r_y}$
- EK_F: $\{{g_v}^{v_k(s)}\}, \{{g_w}^{w_k(s)}\}, \{{g_y}^{y_k(s)}\},$
  $\{{g_v}^{\alpha_v \cdot v_k(s)}\}, \{{g_w}^{\alpha_w \cdot w_k(s)}\}, \{{g_y}^{\alpha_y \cdot y_k(s)}\},$
  $\{g^{s^i}\}, \{{g_v}^{\beta_v \cdot v_k(s)} \cdot {g_w}^{\beta_w \cdot w_k(s)} \cdot {g_y}^{\beta_y \cdot y_k(s)}\}$
- VK_F: $g^1, g^{\alpha_v}, g^{\alpha_w}, g^{\alpha_y}, g^{\gamma}, g^{\beta \cdot \gamma},$
  ${g_y}^{t(s)}, \{{g_v}^{v_k(s)}, {g_w}^{w_k(s)}, {g_y}^{y_k(s)}\}$



	(r_5, r_6)		(r_5, r_6)		(r_5, r_6)
$v_1(r_i)$	(0,1)	$w_1(r_i)$	(0,0)	$y_1(r_i)$	(0,0)
$v_2(r_i)$	(0,1)	$w_2(r_i)$	(0,0)	$y_2(r_i)$	(0,0)
$v_3(r_i)$	(1,0)	$w(r_i)$	(0,0)	$y_3(r_i)$	(0,0)
$v_4(r_i)$	(0,0)	$w_4(r_i)$	(1,0)	$y_4(r_i)$	(0,0)
$v_5(r_i)$	(0,0)	$w_5(r_i)$	(0,1)	$y_5(r_i)$	(0,1)
$v_6(r_i)$	(0,0)	$w_6(r_i)$	(0,0)	$y_6(r_i)$	(1,0)

PINOCCHIO: EVALUATION AND PROOF

- $(y, \pi_y) \leftarrow \text{Compute}(EK_F, u)$:
- Compute $y \leftarrow F(u)$, learn $\{c_i\}$ in the process
- Find $h(x)$ such that $p(x) = t(x) \cdot h(x)$
- π_y: ${g_v}^{v_{mid}(s)}$, ${g_w}^{w_{mid}(s)}$, ${g_y}^{y_{mid}(s)}$, $g^{h(s)}$,
  ${g_v}^{\alpha_v \cdot v_{mid}(s)}$, ${g_w}^{\alpha_w \cdot w_{mid}(s)}$, ${g_y}^{\alpha_y \cdot y_{mid}(s)}$,
  ${g_v}^{\beta \cdot v_{mid}(s)} \cdot {g_w}^{\beta \cdot w_{mid}(s)} \cdot {g_y}^{\beta \cdot y_{mid}(s)}$,
  where $v_{mid}(s) = \sum_{k \in I_{mid}} c_k \cdot v_k(s)$,
  and similarly $w_{mid}(s)$, $y_{mid}(s)$

PINOCCHIO: VERIFICATION

- $\{0,1\} \leftarrow \text{Verify}(VK_F, u, y, \pi_y), \pi_y = g^V, g^W, g^Y, g^H, g^{V'}, g^{W'}, g^{Y'}, g^Z$:
- Divisibility: compute ${g_v}^{v_{IO}(s)} = \prod_{k \in I_{IO}} ({g_v}^{v_k(s)})^{c_k}$ and ${g_w}^{w_{IO}(s)}$, ${g_w}^{w_{IO}(s)}$, then check that $e\left({g_v}^{v_{IO}(s)} \cdot {g_v}^{v_{mid}(s)}, {g_w}^{w_{IO}(s)} \cdot {g_w}^{w_{mid}(s)}\right) = e({g_y}^{t(s)}, g^H) \cdot e({g_y}^{y_{IO}(s)} \cdot {g_y}^{y_{mid}(s)}, g)$
- Linear combinations: $e({g_v}^{V'}, g) = e({g_v}^V, g^{\alpha_v})$, $e({g_w}^{W'}, g) = e({g_w}^W, g^{\alpha_w})$, $e({g_y}^{Y'}, g) = e({g_y}^Y, g^{\alpha_y})$, $e(g^Z, g^\gamma) = e({g_v}^V \cdot {g_w}^W \cdot {g_y}^Y, g^{\beta \cdot \gamma})$

PRIVATE INPUTS

- Some private data D under commitment $C(D)$
- Want to verifiably compute $F(D)$
- Generic solution
  - compute $(F(D), C(D))$ instead
  - then check that $C(D)$ matches

- Pinocchio has direct support for partially private data
- More efficient than the generic scheme above

REFERENCES

- Quadratic programs: Gennaro, Gentry, Parno, Raykova, 2012
  https://eprint.iacr.org/2012/215
- Pinocchio: Parno, Gentry, Howell, Raykova, 2013
  https://eprint.iacr.org/2013/279

- zk-SNARKS
- zk-STARKS
- Bulletproofs

guardtime

THANK YOU

QUESTIONS?

AHTO.TRUU@GUARDTIME.COM
@AHTOTRUU