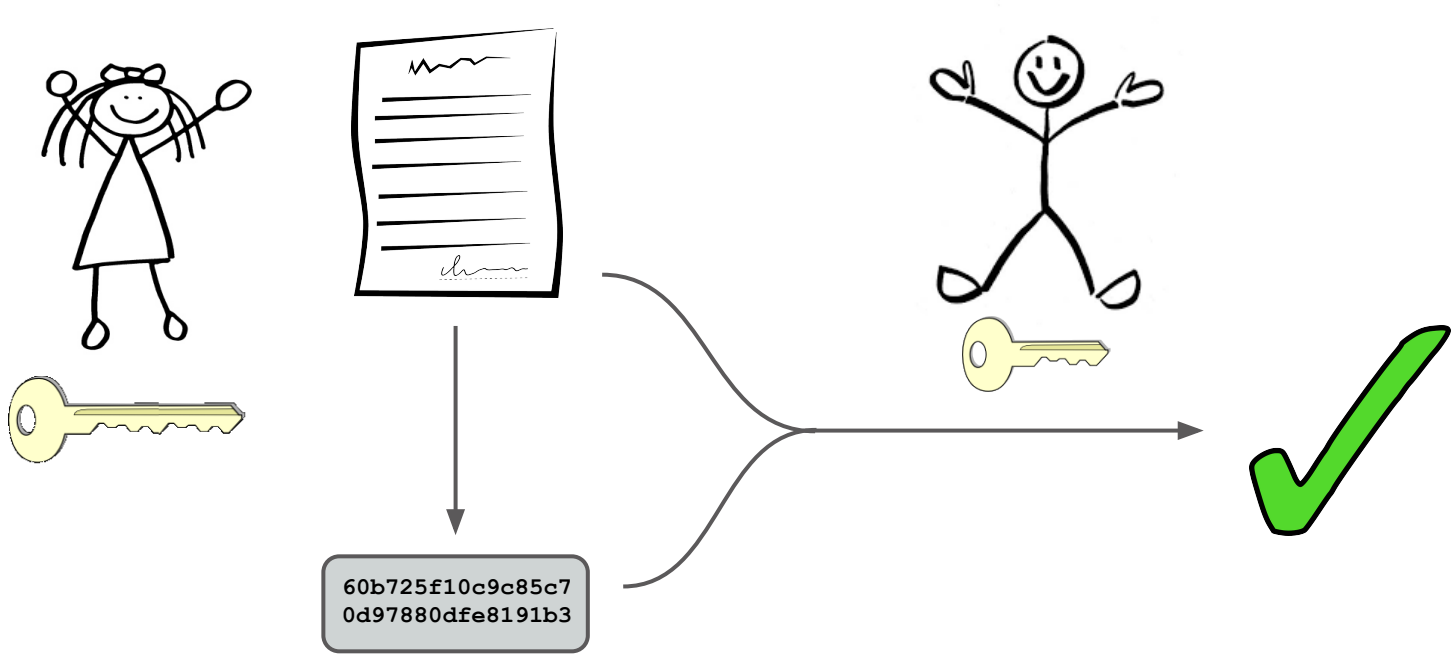
Homomorphic Signature Scheme

By: Dasith, Collin, Sumit

Digital Signatures



Signatures from Public-Key Encryption

- Completeness of Encryption:

  $\mathrm{Dec}_{sk}(\mathrm{Enc}_{pk}(M)) = M$

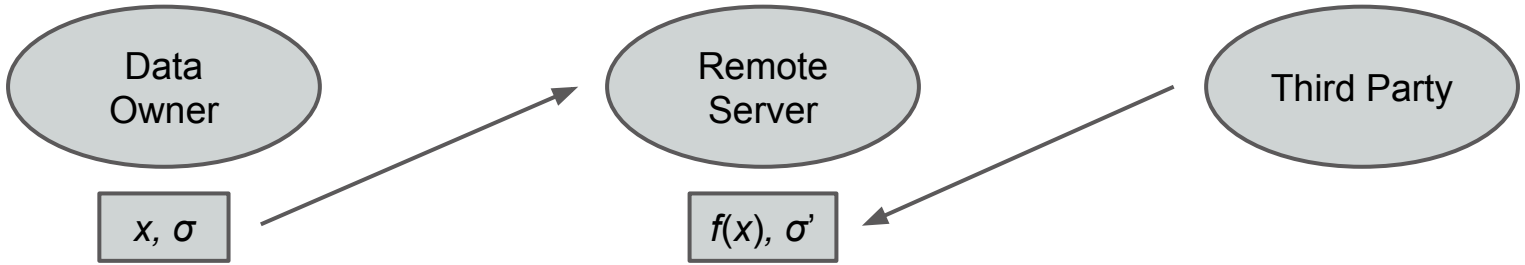
- Why not reverse the order?

  $\sigma = \mathrm{Dec}_{sk}(M)$

  $M = Enc_{pk}(\sigma)$

Homomorphic Signatures

If we have a signature σ for some data x, we want to be able to publicly compute a valid signature σ' on the computation $f(x)$



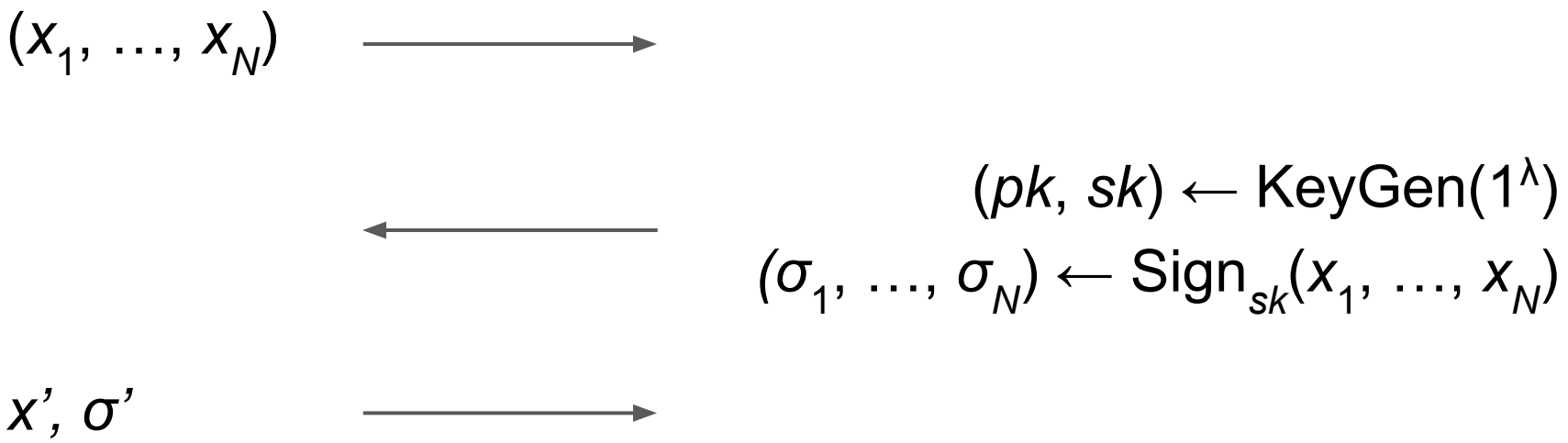
Weakly Homomorphic Signatures

- Note that just as RSA encryption is weakly homomorphic over multiplication, signatures from RSA are weakly homomorphic

- Let σ_1, σ_2 be valid signatures for x_1, x_2, that is, $\sigma_i = x_i^d$

- Then $\sigma_1 \cdot \sigma_2 = x_1^d \cdot x_2^d = (x_1 \cdot x_2)^d$ is a valid signature for the message $x_1 \cdot x_2$

Security of Signature Schemes



Adversary wins if $x' \notin (x_1, \ldots, x_N)$
but σ' is a valid signature for x'

Security of Homomorphic Signatures

- If our scheme is homomorphic over a binary operation $\odot : M \times M \rightarrow M$ this notion of security is not good enough
- Adversary is able to generate valid signatures for $x_1 \odot x_2$, $(x_3 \odot x_4) \odot x_5$ and so on
- That is, any message in $\text{span}_{\odot}(x_1, \ldots, x_N)$

Security of Homomorphic Signatures

$(x_1, \ldots, x_N)$ →

← $(pk, sk) \leftarrow \text{KeyGen}(1^\lambda)$

← $(\sigma_1, \ldots, \sigma_N) \leftarrow \text{Sign}_{sk}(x_1, \ldots, x_N)$

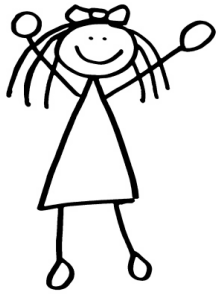
x', σ' →

Adversary wins if $x' \notin \text{span}_{\odot}(x_1, \ldots, x_N)$
but σ' is a valid signature for x'

Homomorphic Signature Scheme

The big picture is given a signature of some message, another party will be able to compute a valid signature of the sub-message, or a subset of the message also known as redaction. This idea can be represented in a scenario with two people Alice and Bob.

Scenario



Alice

MESSAGE
x1x2x3x4x5

Signed
Management

Alice wants to show another party, lets say Bob, but Bob is not allowed to see certain parts of the message.

Bob

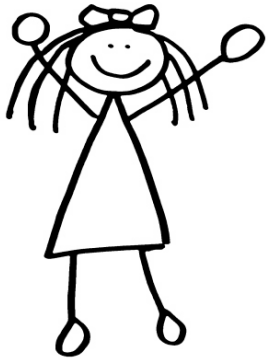
Scenario Continued

MESSAGE
x1x2x3x4x5

Signed
Management

The Message contains x1,x2, x3, ..., ,xn, which represents the words or letters in the message. Also known as redacting the message

Scenario Continued



MESSAGE
x1x2x3x4x5

Signed
Management

MESSAGE
##x3x4x5

Signed
Management

Alice

If Bob is not allowed to see the word or characters at x1 and x2, Alice would just replace those two elements from the message with symbols

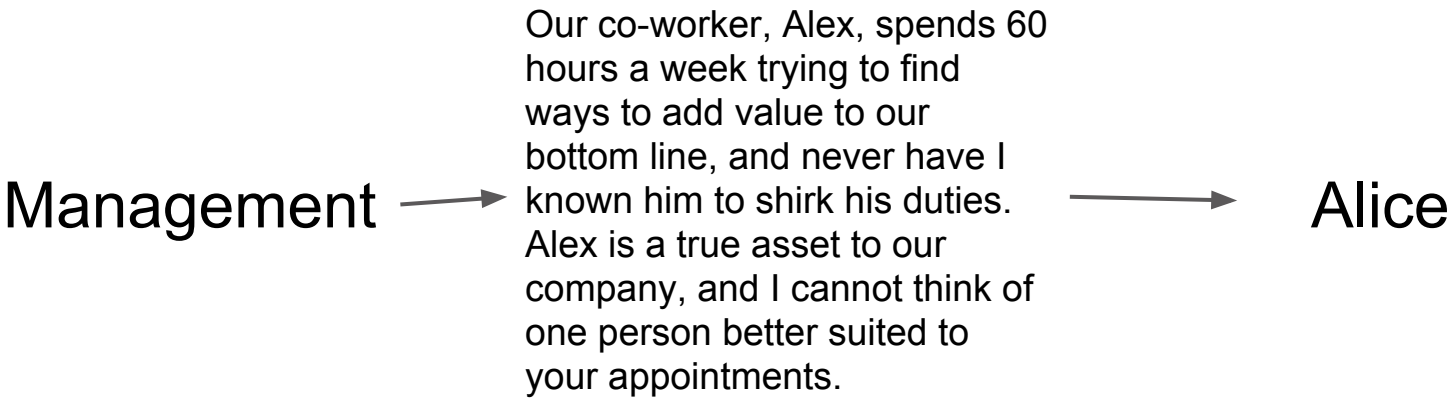
Bob

Scenario Continued

Why would we use symbols to replace the character or word instead of simply just deleting it entirely?

Scenario Continued

By showing the positions where the elements have been deleted is that the meaning of the message could change.



Scenario

Alice could remove some of the words to change the message's meaning

Our co-worker, Alex, spends 60 hours a week trying to find ways to add value to our bottom line, and never have I known him to shirk his duties. Alex is a true asset to our company, and I cannot think of one person better suited to your appointments.

Our co-worker, Alex, spends 60 hours a week trying to shirk his duties, and I can think of one person better suited to your appointments.

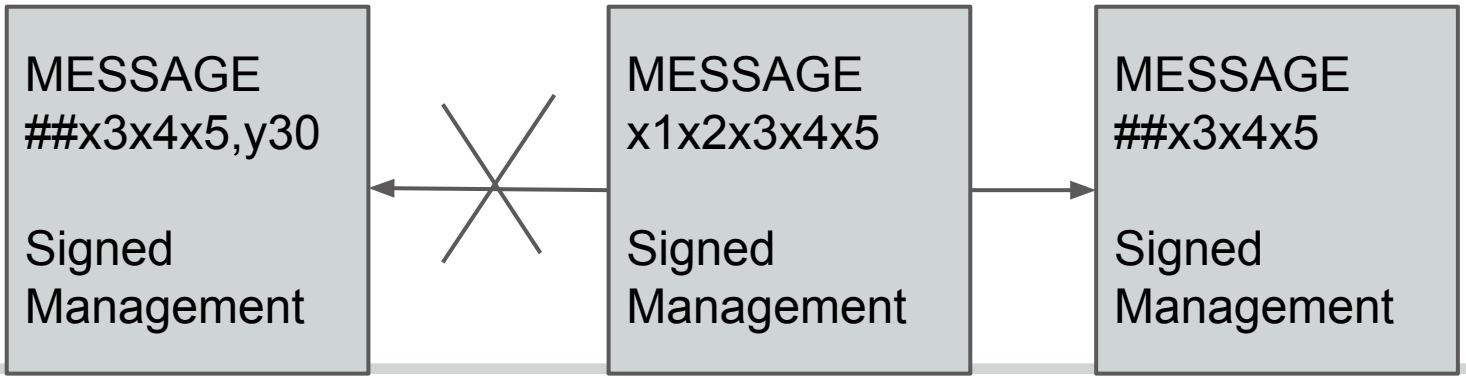
Scenario Continued

Our co-worker, Alex, spends 60 hours a week trying to find ways to add value to our bottom line, and never have I known him to shirk his duties. Alex is a true asset to our company, and I cannot think of one person better suited to your appointments.

Our co-worker, Alex, spends 60 hours a week trying to #### #### ## ### ##### ## ### ###### #### ### ##### #### ##### ### ## shirk his duties #### ## # #### ##### ## ### ######## and I can### think of one person better suited to your appointments.

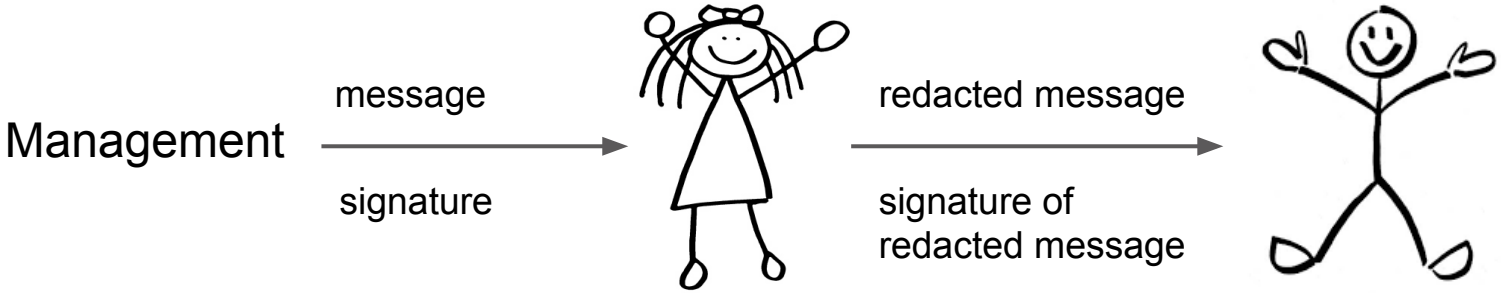
Scenario Continued

The idea of this paper is to construct an algorithm which will verify whether the redacted message is actually a sub-message of the original message.

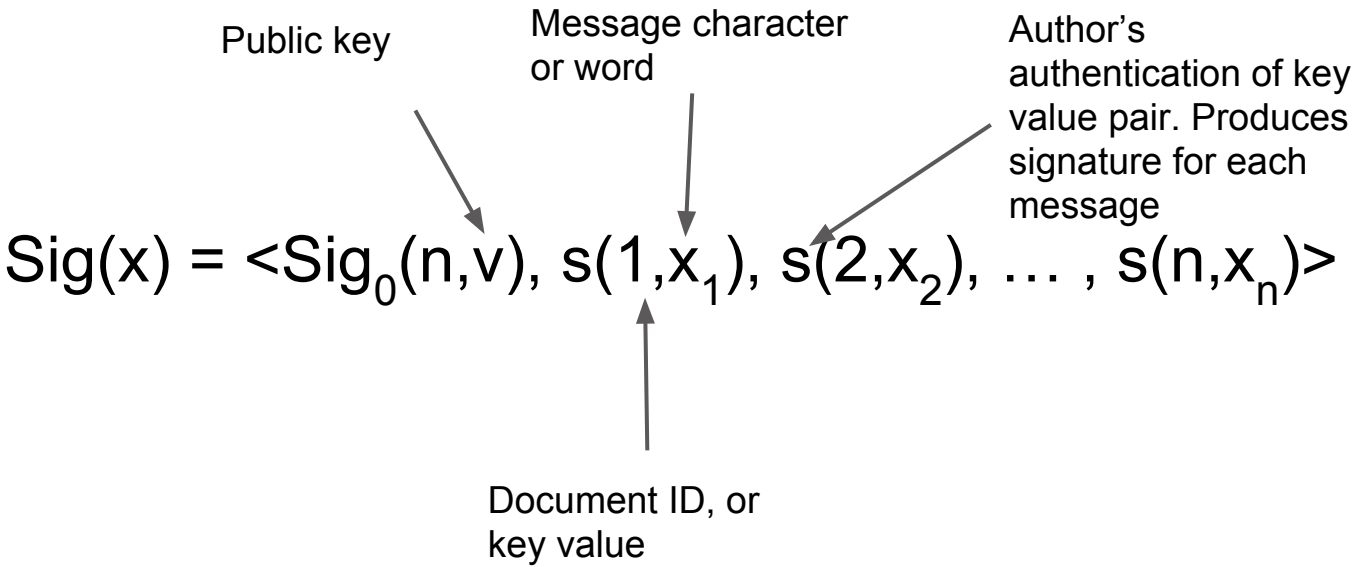


Scenario Continued

When Alice sends Bob the redacted message, Bob might want to make sure that this is the original message and would want to verify the authenticity of it.



Trivial scheme for Redactable Sig.



Redaction for Trivial Solution

$$Sig(x) = \langle Sig_0(n,v),\ s(1,x_1),\ s(2,x_2),\ \ldots\ ,\ s(n,x_n) \rangle$$

If you want to redact the character from position x_1

$$Sig(x) = \langle Sig_0(n,v),\ s(2,x_2),\ \ldots\ ,\ s(n,x_n) \rangle$$

Will not reveal the redacted parts of message but it will reveal locations.

Issue with Trivial

$$Sig(x) = \langle Sig_0(n,v),\ s(1,x_1),\ s(2,x_2),\ \ldots\ ,\ s(n,x_n) \rangle$$

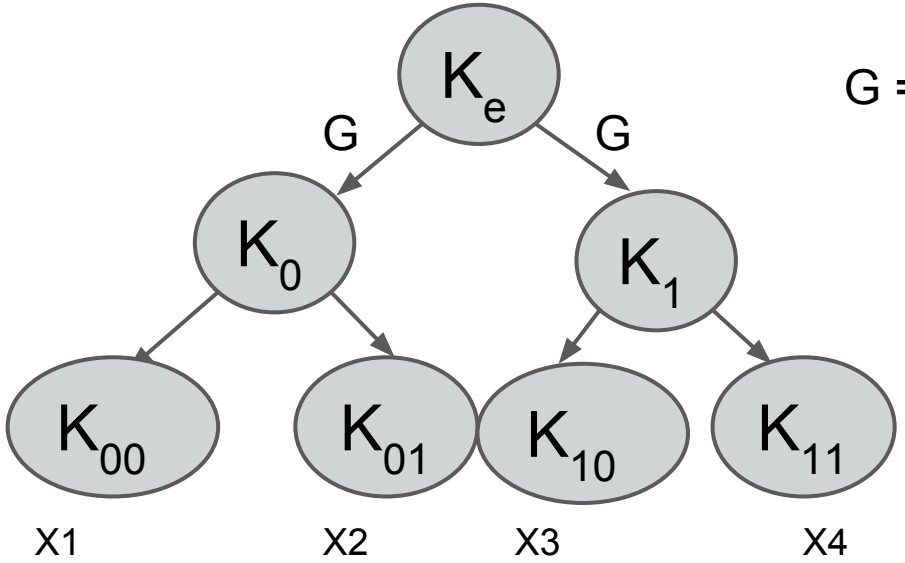
Sig(x) is very long. Consider that each “s” will produce a m bit signature, and the signature applies to each letter of the message (n), so the length of Sig(x) = mn + O(1) ← an arbitrary constant. Much longer than the length of the message: n

Tree Construction Main Idea

1) Construct a tree where each leaf is associated with a letter in the message.
2) Compute a hash value for each leaf. Then climb the tree combining all the signed letter.
3) Sign the root of the tree and compute Sig(x)

Expansion of the Tree

A randomly generated K_e is the root node and a given function G is used to recursively create the tree.

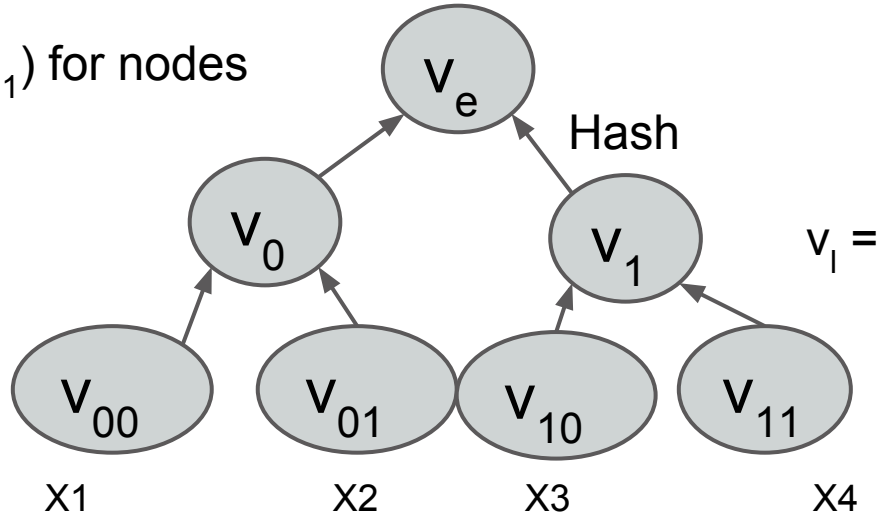


$G = <K_{n0}, K_{n1}>$

Hashing of Tree

Take all the key, value pairs and apply a hash function to them. Analogous to $s(k,x_n)$ from trivial. Create a new tree.

$v_n = Hash(1,v_{n0},v_{n1})$ for nodes

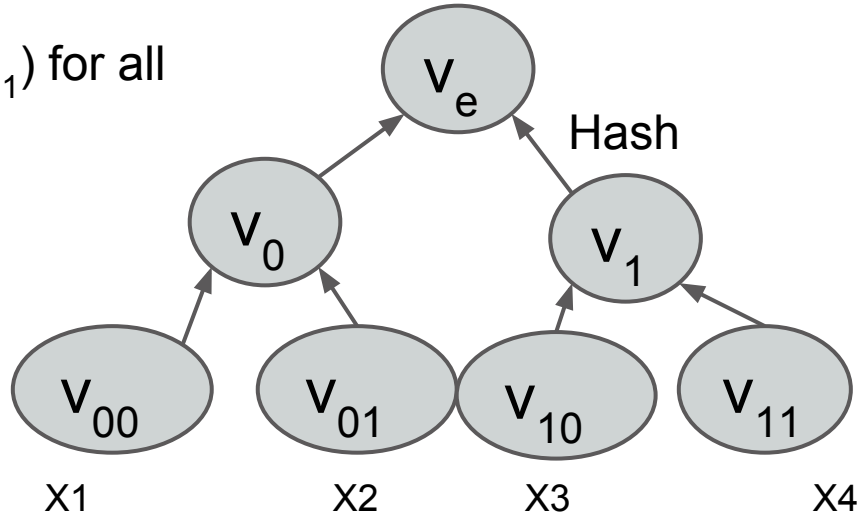


$v_l = Hash(0,x_l,K_l)$ for leaves

Signing the Tree

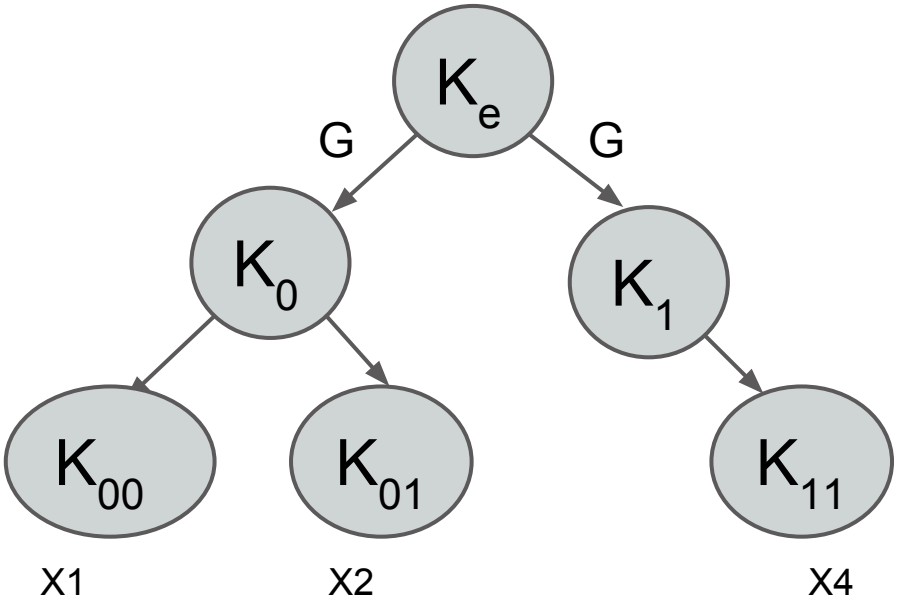
Use $Sig_0(v_e)$ and K_e to check Sig(x).

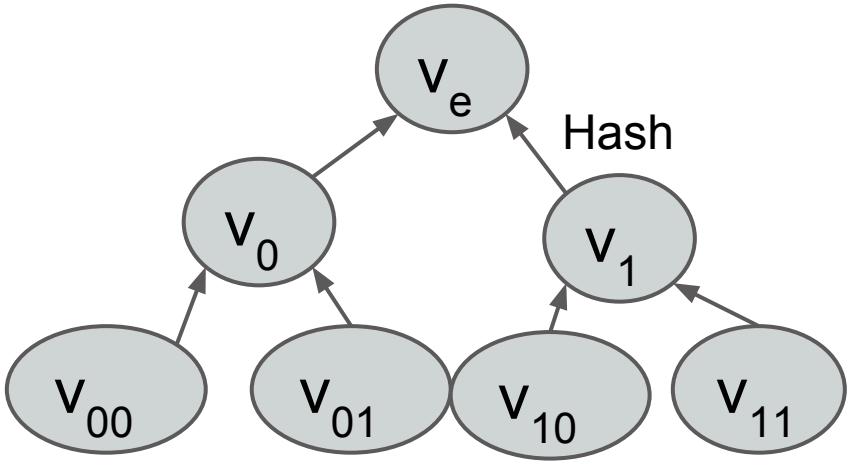
$v_e = Hash(1, v_{n0}, v_{n1})$ for all nodes



Redaction with Tree Structure

To calculate Signature of redaction, we need to remove the character value along with the key. So if we want to keep character x3 hidden we would remove it from the tree.



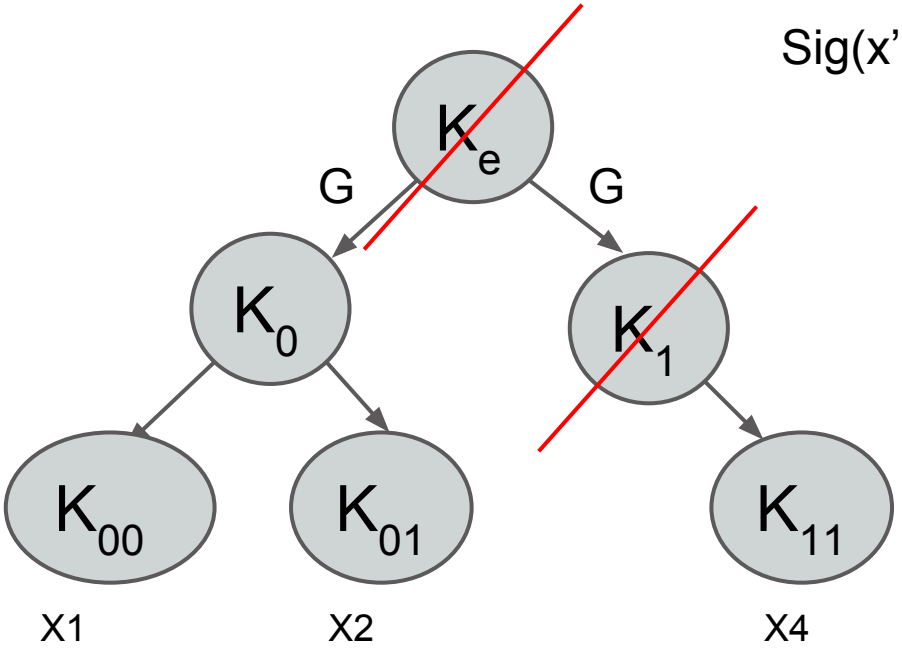


Issue With Only Removing Character

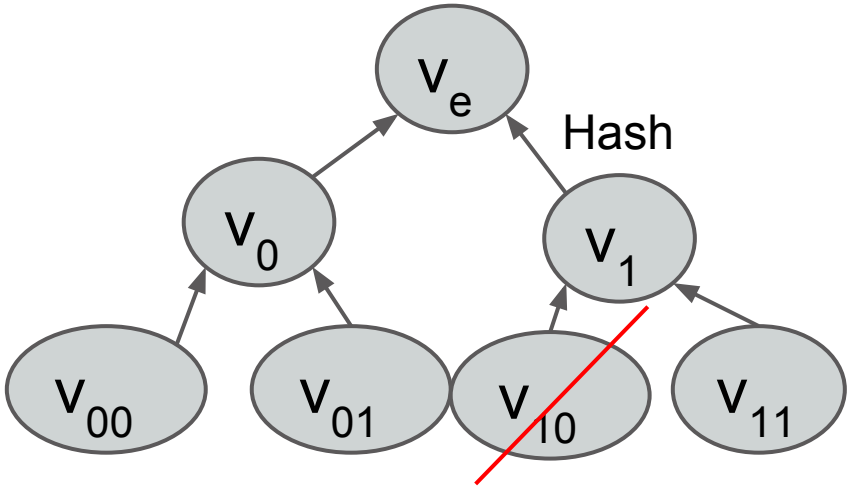
Since the K_e and recursive G function is given to the other party, they can generate all the keys associated with the different values. Since we'll need all the "v" values (Hash(0, x_l,K_l)), the other party can iterate through all the values of x_l to see which key represents that character and know the identity of that redacted character.

Solution: Hide any ancestor keys that can allow the formation of the redacted character's key

Redaction Tree Construction ver. 2



$Sig(x') = <k_0, k_{11}, v_{10}, Sig_0(v_e)>$



Runtime of Tree Redaction

Summing it all up:

Reason we went over this construction is because it is shorter than the trivial construction. Reminder: the trivial construction was length n*m + some constant.

Unredacted Tree Construction: m + m’

Tree Construction: m + O(s*m’ log(n * n’))

m = Sig_0 m’ = hash value and keys

n = length of message s = removed segments

n’ = length longest removed segment

Fully Homomorphic Signatures

- Given data $x = (x_1, \ldots, x_N)$ and associated signatures $(\sigma_1, \ldots, \sigma_N)$, we want to be able to compute a signature σ^* for the result of some computation over x, $y = g(x_1, \ldots, x_N)$

Fully Homomorphic Signatures

- $(pk, sk) \leftarrow \text{KeyGen}(1^{\lambda}, 1^{N})$
- $(\sigma_1, \ldots, \sigma_N) \leftarrow \text{Sign}_{sk}(x_1, \ldots, x_N)$
- $\sigma^* = \text{Eval}_{pk}(g, ((x_1, \sigma_1), \ldots, (x_N, \sigma_N)))$
- $\text{Verify}_{pk}(g, y, \sigma^*)$

Security Game

$(x_1, \ldots, x_N)$ →

← $(pk, sk) \leftarrow \text{KeyGen}(1^\lambda, 1^N)$

← $(\sigma_1, \ldots, \sigma_N) \leftarrow \text{Sign}_{sk}(x_1, \ldots, x_N)$

g, y', σ' →

Adversary wins if $y' \neq g(x_1, \ldots, x_N)$ but $\text{Verify}_{pk}(g, y', \sigma') =$ `accept`

Construction (Wichs, 2014)

- Introduce Homomorphic Trapdoor Functions
- Construct *leveled* signature scheme
  - Fix maximal degree of polynomial
  - Bounded data

Homomorphic Trapdoor Functions

- One direction is easy to compute: $f_{pk}(x, u) = v$
- Trapdoor for inversion: $\text{Inv}_{sk}(x, v)$
- Given values u_i, x_i, v_i and a function g, construct u^*, v^* so that $f_{pk}(g(x_1, \ldots, x_N), u^*) = v^*$

Homomorphic Trapdoor Functions

- $(pk, sk) \leftarrow \text{KeyGen}(1^{\lambda})$
- Given u_i, x_i, set $v_i = f_{pk}(x_i, u_i)$, $1 \le i \le N$
- Trapdoor: $f_{pk}(x, \text{Inv}_{sk}(x, v)) = v$
- $u^* = \text{Eval}^{\text{in}}_{pk}(g, (x_1, u_1), \ldots, (x_N, u_N))$
  $v^* = \text{Eval}^{\text{out}}_{pk}(g, v_1, \ldots, v_N)$
  $f_{pk}(g(x_1, \ldots, x_N), u^*) = v^*$

Security of HTDFs

- One-wayness: it should be hard to find some pre-image u such that $f_{pk}(x, u) = v$ for given x, v

- Claw-freeness: it should be hard to find inputs u, u' and $x \neq x$' such that $f_{pk}(x, u) = f_{pk}(x', u')$

Signatures from HTDFs

$(pk, sk) \leftarrow \text{KeyGen}(1^{\lambda}, 1^{N})$

- Choose public parameters $prms = (v_1, \ldots, v_N)$ uniformly at random
- Generate $(pk', sk') \leftarrow \text{KeyGen}^{\text{HTDF}}(1^{\lambda})$
- Return $pk = (prms, pk')$, $sk = (prms, sk')$

Signatures from HTDFs

$(\sigma_1, \ldots, \sigma_N) \leftarrow \mathrm{Sign}_{sk}(x_1, \ldots, x_N)$

- Generate a preimage u_i such that $f_{pk'}(x_i, u_i) = v_i$
- $u_i \leftarrow \mathrm{Inv}_{sk'}(x_i, v_i)$
- Return $\sigma_i = u_i$

- Then $f_{pk}(x_i, u_i) = v_i$

Signatures from HTDFs

$\sigma^* = \text{Eval}_{pk}(g, ((x_1, \sigma_1), \ldots, (x_N, \sigma_N)))$

- Simply return
  $\sigma^* = u^* = \text{Eval}^{\text{in}}_{pk'}(g, (x_1, \sigma_1), \ldots, (x_N, \sigma_N))$

Signatures from HTDFs

$\text{Verify}_{pk}(g, y, \sigma^*)$

- Compute $v^* = \text{Eval}^{\text{out}}_{pk'}(g, v_1, \ldots, v_N)$
- Check whether $f_{pk'}(y, \sigma^*) = v^*$
- Accept or reject accordingly

Security of Signature Scheme

- Assume an attacker has found some values (g, y', σ') where $y' \neq y$, but $\text{Verify}_{pk}(g, y', \sigma')$ accepts the signature σ' as valid
- The attacker can also compute (g, y, σ^*), where $\sigma^* = \text{Eval}_{pk}(g, ((x_1, \sigma_1), \ldots, (x_N, \sigma_N)))$ so that σ^* is the valid signature for y
- Then $f_{pk}(y, \sigma) = f_{pk}(y', \sigma')$, which breaks the claw-freeness of the HTDF

Background on Lattice Problems

- Short Integer Solutions (SIS)
- Given a random integer matrix A, we want to find an integer vector u with $A \cdot u = 0 \pmod{q}$
- u cannot be the zero vector
- $||u|| < \beta$ for some upper bound β

Background on Lattice Problems

- Inhomogeneous Short Integer Solutions (ISIS)
- Given a random integer matrix A and some integer matrix V, we want to find an integer matrix U with $A \cdot U = V \pmod{q}$
- Equivalent to SIS
- Hard in average case, but there are trapdoors
- There is a matrix G for which ISIS is easy

Constructing HTDFs

$(pk, sk) \leftarrow \text{KeyGen}(1^{\lambda})$

- Sample a random matrix A along with its SIS trapdoor td
- Return $pk = A$, $sk = td$

Constructing HTDFs

$v = f_{pk}(x, u)$

- Let G be a matrix for which SIS is easy
- Define $f_A(x, U) \coloneqq A \cdot U - x \cdot G$

Constructing HTDFs

$f_{pk}(x, \mathrm{Inv}_{sk}(x, v)) = v$

- Use the trapdoor for $A = pk$ to find a solution matrix U to the ISIS problem $A \cdot U = V + x \cdot G$
- Return $\mathrm{Inv}_{td,x}(V) = U$
- $f_A(x, \mathrm{Inv}_{td}(x, V)) = A \cdot U - x \cdot G = V + x \cdot G - x \cdot G = V$

Homomorphic Properties of HTDFs

$u^* = \mathrm{Eval}^{\mathrm{in}}_{pk}(g, (x_1, u_1), \ldots, (x_N, u_N))$

$v^* = \mathrm{Eval}^{\mathrm{out}}_{pk}(g, v_1, \ldots, v_N)$

- Addition: let $g(x_1, x_2) = x_1 + x_2$
- Set $U^* = U_1 + U_2$, $V^* = V_1 + V_2$
- $f_{A,x_1 + x_2}(U^*) = \mathrm{A}\cdot\mathrm{U}^* - (x_1 + x_2)\cdot G$
  $= (A\cdot U_1 - x_1\cdot\mathrm{G}) + (A\cdot U_2 - x_2\cdot\mathrm{G}) = V^*$

Homomorphic Properties of HTDFs

$u^* = \text{Eval}^{\text{in}}_{pk}(g, (x_1, u_1), \ldots, (x_N, u_N))$

$v^* = \text{Eval}^{\text{out}}_{pk}(g, v_1, \ldots, v_N)$

- Multiplication: let $g(x_1, x_2) = x_1 \cdot x_2$
- Find a matrix R such that $G \cdot R = -V_2$
- Set $U^* = x_2 \cdot U_1 + U_2 \cdot R$, $V^* = V_2 \cdot R$

Homomorphic Properties of HTDFs

$$
\begin{aligned}
f_{A,x_1 \cdot x_2}(U^*) &= A \cdot U^* - (x_1 \cdot x_2) \cdot G \\
&= A \cdot (x_2 \cdot U_1 + U_2 \cdot R) - (x_1 \cdot x_2) \cdot G \\
&= x_2 \cdot (A \cdot U_1 - x_1 \cdot G) + A \cdot U_2 \cdot R \\
&= x_2 \cdot V_1 + (V_2 + x_2 \cdot G) \cdot R \\
&= x_2 V_1 + V_2 \cdot R - x_2 \cdot V_1 = V_2 R = V^*
\end{aligned}
$$

Security of HTDF Construction

- Assume we can find a claw (U_0, U_1) such that $f_{A,0}(U_0) = f_{A,1}(U_1)$
- That is, $A \cdot U_0 - 0 \cdot G = A \cdot U_1 - 1 \cdot G$
- Or $A \cdot (U_1 - U_0) = G$
- But we can sample a short r such that $G \cdot r = 0$
- Then $u = (U_1 - U_0) \cdot r$ is a short vector such that $A \cdot u = 0$, which breaks SIS

Noise Growth

- With SIS and ISIS, we are looking for *short* solution matrices
- When we homomorphically evaluate new signatures, we are increasing the size of these matrices, $\beta \rightarrow (m + 1)\beta \rightarrow \ldots \rightarrow (m + 1)^d\beta$
- Choose a maximum degree d for functions

Construction (Gorbunov & Vaikuntanathan, 2014)

- Similar to the Wichs' construction
- Idea for multiple datasets: tags
- Explicit construction for circuit representation of a function

Multiple Datasets

- Tag $t \in \{0,1\}^{\lambda}$ associated with each dataset
- New security game:
  - Adversary can make calls to an oracle $O_{sk}(.)$
  - *q calls*
  - $\{\vec{x_j}\}_{j \in [q]}$ queried datasets; $\{\vec{\sigma_j}, t_j\}_{j \in [q]}$ replies
- Adv. wins if $Verify(pk, t^*, x^*, \sigma^*, c^*) = 1$ and:
  - $t^* \neq t_j$ for all $j \in [q]$ **OR**
  - $t^* = t_j$ for some $j \in [q]$ but $x^* \neq c^*(\vec{x_j})$

Definition

- Homomorphic signature scheme for the class of circuits C
- $HS = (Setup, Sign, Eval, Verify)$

Definition (cont.)

- $Setup(1^{\lambda}, 1^{n}) \rightarrow (pk, sk)$
- $Sign(sk, t, i, x) \rightarrow \sigma$
- $Eval(pk, t, (x_1, \ldots, x_n), (\sigma_1, \ldots, \sigma_n), c) \rightarrow \sigma'$
- $Verify(pk, t, x', \sigma', c) \rightarrow \{0,1\}$

Circuit Representation

- Reduce to NAND gates
- Let C_λ be a collection of Boolean circuits with at most n inputs
- For $c \in C_\lambda$
- Input wires indexed *1* to *n*
- Internal wires indexed *n+1* to $|c|$
- Each gate is a tuple *(u,v,w)*

$Setup(1^{\lambda}, 1^{n}, 1^{d})$

- Sample random matrices: $\boldsymbol{A}$, $\{\boldsymbol{D}_i\}_{i \in [n]}$ in $\mathbb{Z}_q^{n \times m}$
- Sample two matrices with associated trapdoors: $(\boldsymbol{A}^*, \boldsymbol{T}_{\boldsymbol{A}^*})$ and $(\boldsymbol{B}, \boldsymbol{T}_{\boldsymbol{B}})$
- Output public key and secret key:

  $pk = (\boldsymbol{A}, \boldsymbol{A}^*, \boldsymbol{B}, \boldsymbol{T}_{\boldsymbol{B}}, \{\boldsymbol{D}_i\}_{i \in [n]})$

  $sk = \boldsymbol{T}_{\boldsymbol{A}^*}$

Sign(sk, t, i, x)

- Dataset lattice: $\boldsymbol{A}_t := [\boldsymbol{A}^* | \boldsymbol{A}+t\boldsymbol{B}]$
- Sample $[\boldsymbol{R}_2 | \boldsymbol{R}_1]$ s.t.:

$$\mathbf{A}_t \begin{matrix} \boldsymbol{R}_2 \\ \boldsymbol{R}_1 \end{matrix} = \boldsymbol{A}^*\boldsymbol{R}_2 + (\boldsymbol{A}+t\boldsymbol{B})\boldsymbol{R}_1 = \boldsymbol{D}_i + x\boldsymbol{B}$$

- Output signature $\sigma = [\boldsymbol{R}_2 | \boldsymbol{R}_1]$

Eval(pk, t, $\vec{u}$, $\vec{\sigma}$, c)

- Compute homomorphic signature recursively
- For each wire *i*, let $\boldsymbol{D}_i$ be the “public key” associated with the wire
- Gate *g=(u,v,w)* carrying inputs values *a,b*
- Signatures for inputs *u,v* are $\boldsymbol{R}_u, \boldsymbol{R}_v$
- $\boldsymbol{A}_t\mathbf{R}_u = \mathbf{D}_u + a\mathbf{B}$
- $\boldsymbol{A}_t\mathbf{R}_v = \mathbf{D}_v + b\mathbf{B}$

Eval(pk, t, $\vec{u}$, $\vec{\sigma}$, c) (cont.)

- Define public key for output wire as

  $\boldsymbol{D}_w := \boldsymbol{D}_v \boldsymbol{D}_u^{\sim} - \boldsymbol{B}$

- $\boldsymbol{D}_u^{\sim} \in \mathbb{Z}^{m \times m}$ s.t. $\boldsymbol{B}\boldsymbol{D}_u^{\sim} = \mathbf{D}_u$
- Compute homomorphic signature:

  $\boldsymbol{R}_w = \boldsymbol{R}_v \boldsymbol{D}_u^{\sim} - b\boldsymbol{R}_u$

Verify(pk, t, x, σ, c)

- Take $\mathbf{D}_c = \mathbf{D}_{|c|}$ (public key for circuit) and verify that:

  $\mathbf{A}_t\mathbf{R} = \mathbf{D}_c + x\mathbf{B} \bmod q$

Correctness

- Want to show that

  $\boldsymbol{A}_t\boldsymbol{R} = \boldsymbol{D}_c + c(\vec{x})\boldsymbol{B}$

- Idea: signature for each gate output is valid, entire circuit must be valid

Correctness (cont.)

- For $g = (u,v,w)$ carrying inputs a,b
- $\boldsymbol{A}_t\boldsymbol{R}_w = \boldsymbol{A}_t(\boldsymbol{R}_v\boldsymbol{D}_u{}^{\sim} - b\boldsymbol{R}_u)$

$$= \boldsymbol{A}_t\boldsymbol{R}_v\boldsymbol{D}_u{}^{\sim} - b\boldsymbol{A}_t\boldsymbol{R}_u$$

$$= (\boldsymbol{D}_v + b\boldsymbol{B})\boldsymbol{D}_u{}^{\sim} - b(\boldsymbol{D}_u + a\boldsymbol{B})$$

$$= \boldsymbol{D}_v\boldsymbol{D}_u{}^{\sim} + b\boldsymbol{D}_u - b\boldsymbol{D}_u - ab\boldsymbol{B})$$

$$= \boldsymbol{D}_v\boldsymbol{D}_u{}^{\sim} - ab\boldsymbol{B}$$

$$= \boldsymbol{D}_w + (1-ab)\boldsymbol{B}$$

$$= \boldsymbol{D}_w + (a \text{ NAND } b)\boldsymbol{B}$$

Correctness (cont.)

- NAND is computed correctly, signature for every output can be computed

The End