How to Use (Plain) Witness Encryption: Flexible Broadcast, Registered ABE, and More

Cody Freitag, Brent Waters, and David Wu

May 2023

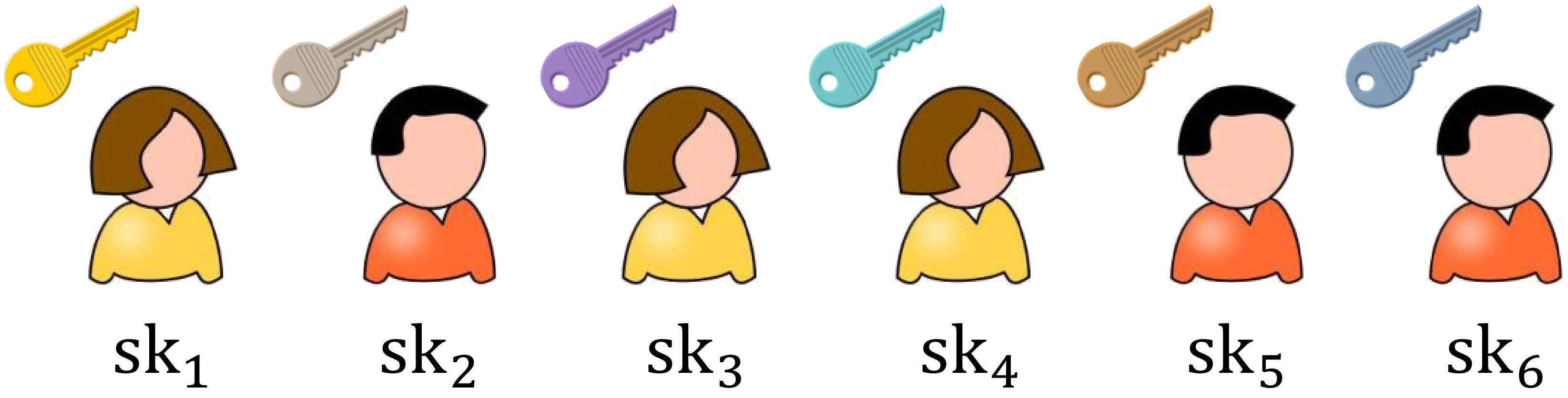
Broadcast Encryption

[FN93]

message m

$S = \{1,3,6\}$

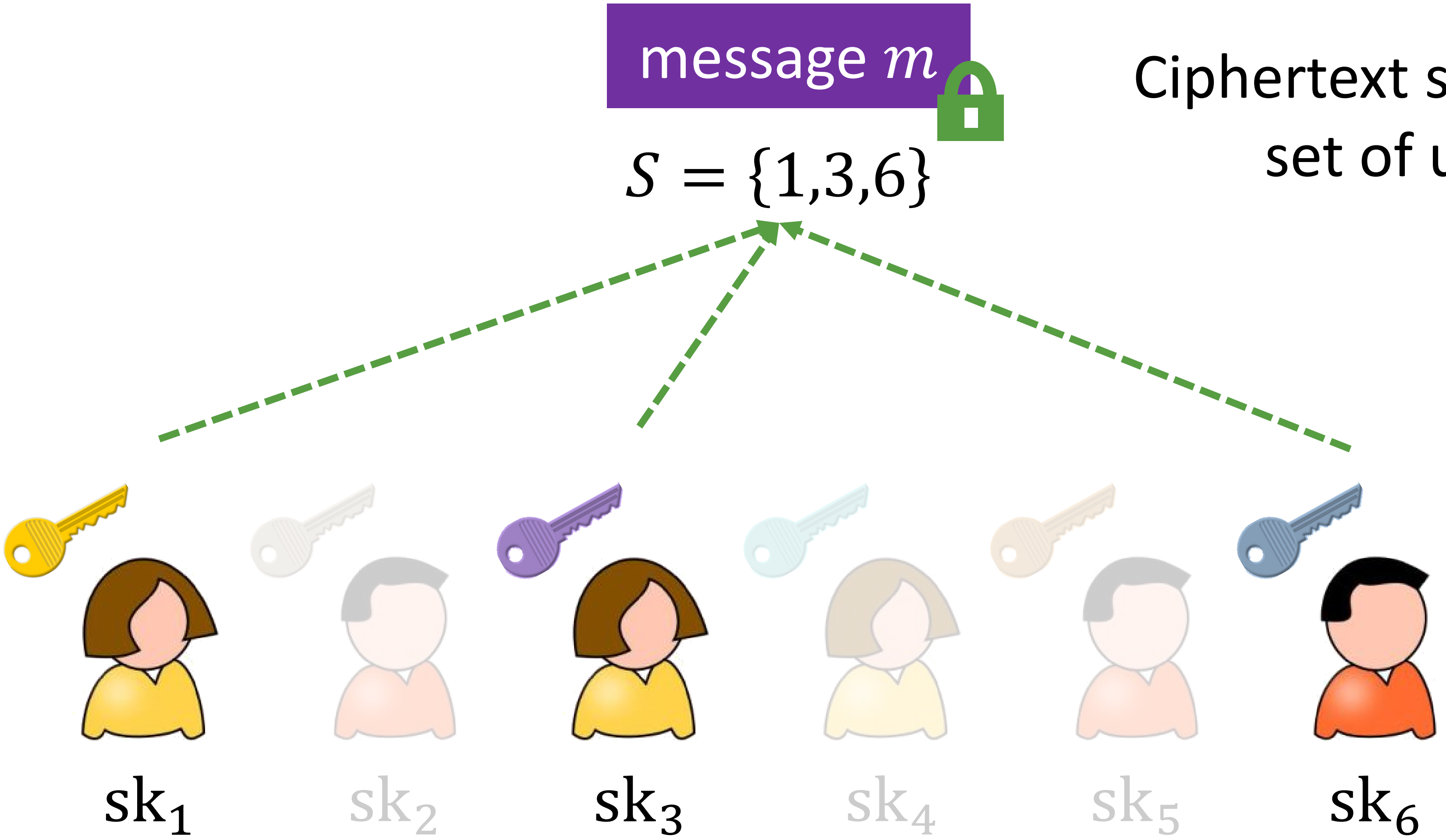
Ciphertext specifies a set of users



Broadcast Encryption

[FN93]

Functionality: Users in the set can decrypt



Ciphertext specifies a set of users

Broadcast Encryption

[FN93]

Functionality: Users in the set can decrypt

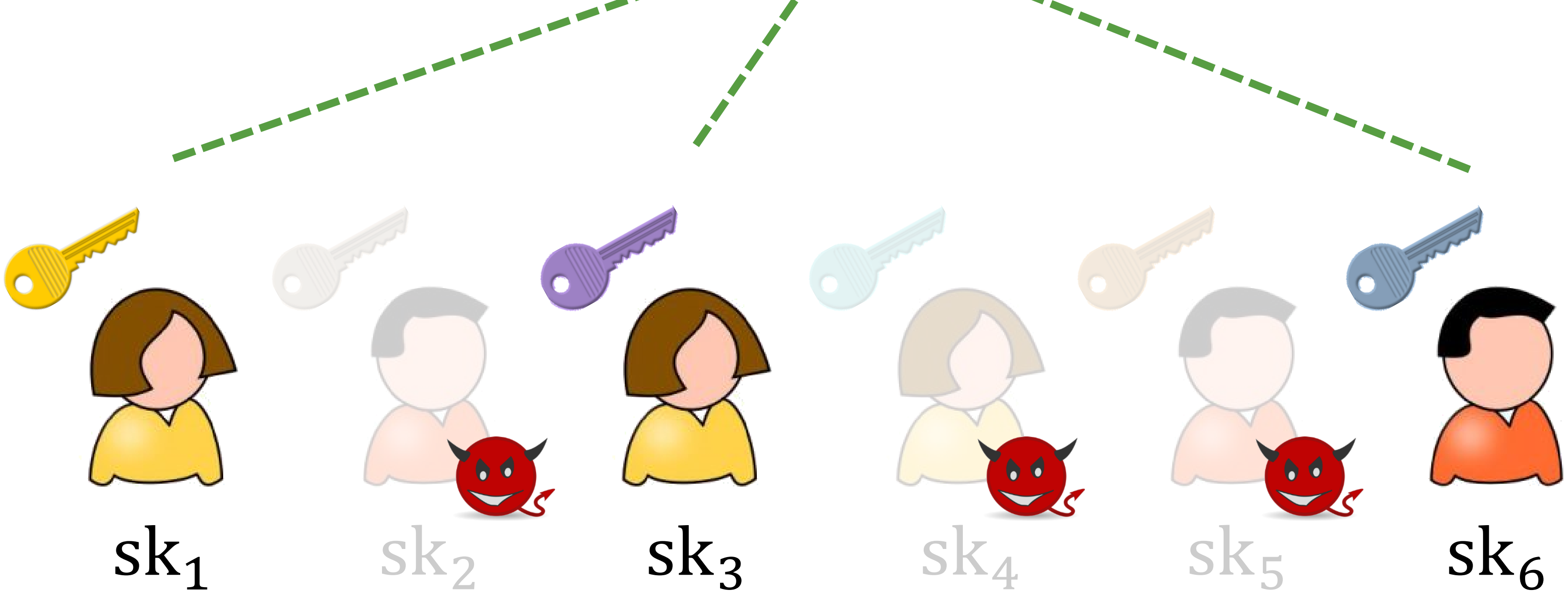
Security: Users outside the set learn nothing about message (even if they collude)

message m

$S = \{1,3,6\}$

Ciphertext specifies a set of users

Efficiency: $|\text{ct}| = |m| + \text{poly}(\lambda, \log|S|)$



Broadcast Encryption

[FN93]

Functionality: Users in the set can decrypt

Security: Users outside the set learn nothing about message (even if they collude)

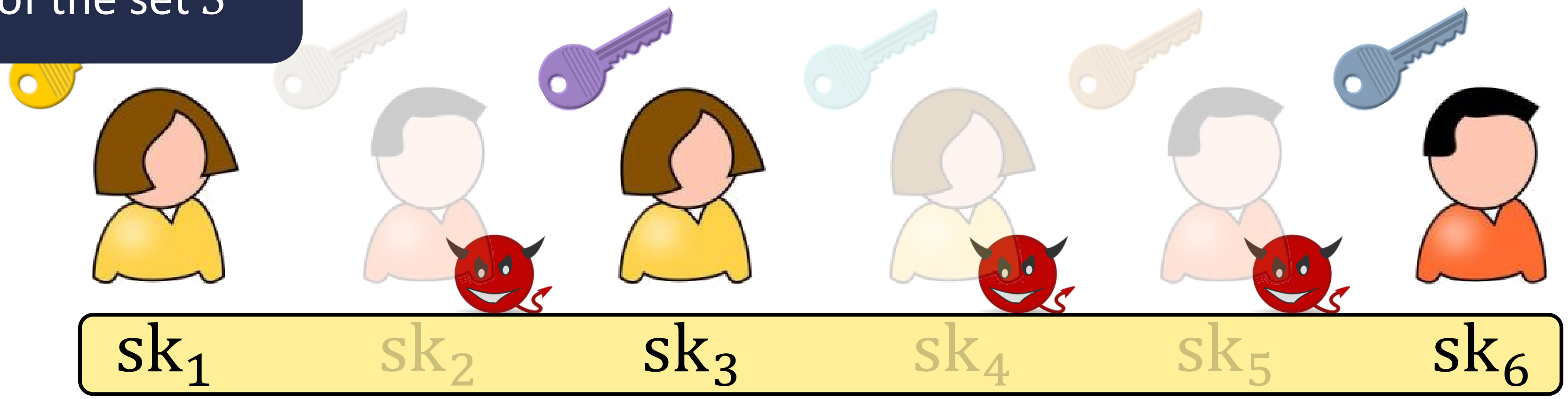
Efficiency: $|\mathrm{ct}| = |m| + \mathrm{poly}(\lambda, \log|S|)$

message m

$S = \{1,3,6\}$

Ciphertext specifies a set of users

Note: decryption requires knowledge of the set S



Where do the secret keys come from?

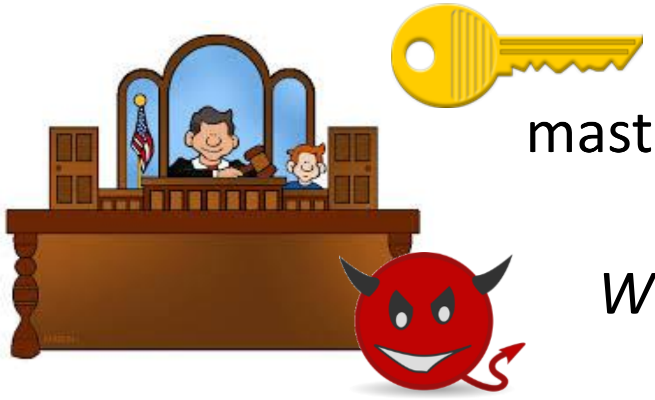
Broadcast Encryption

[FN93]

Central **trusted** authority generates keys

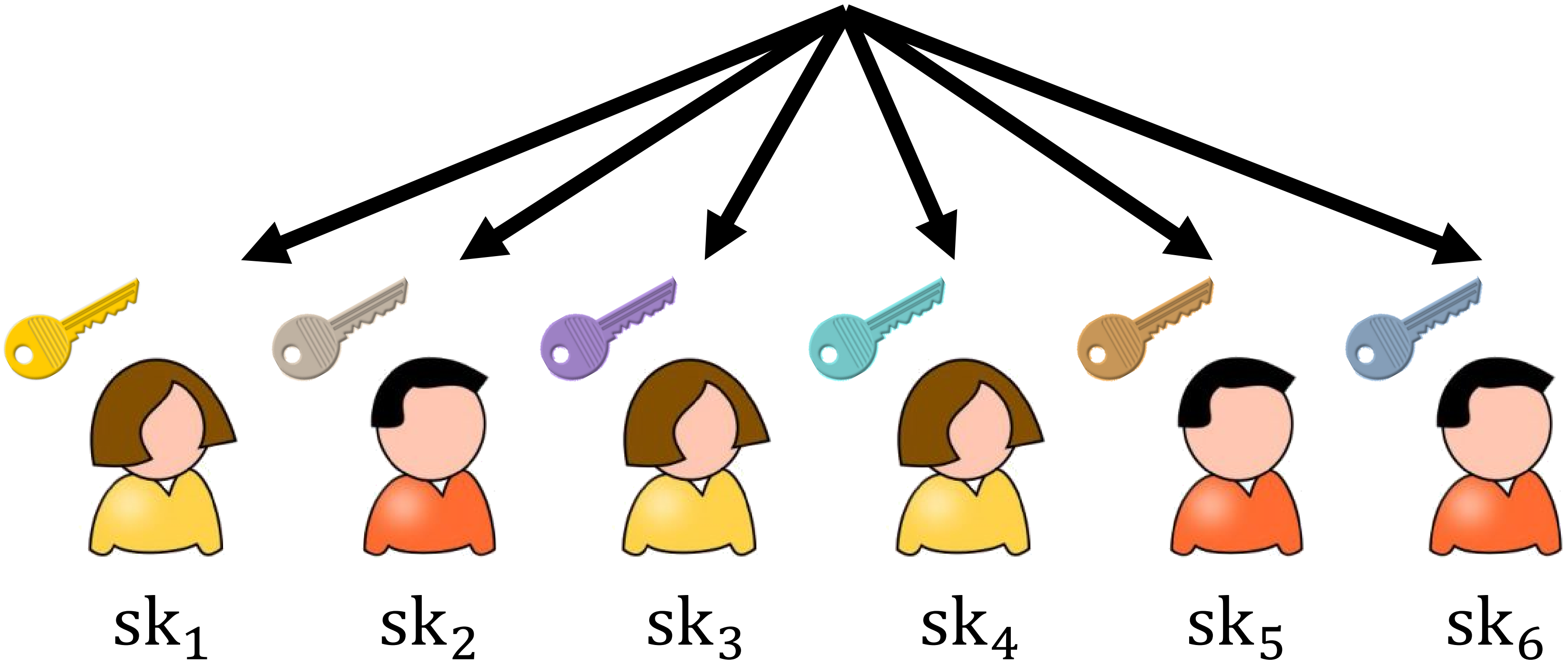
Built-in **key escrow**

Central point of failure

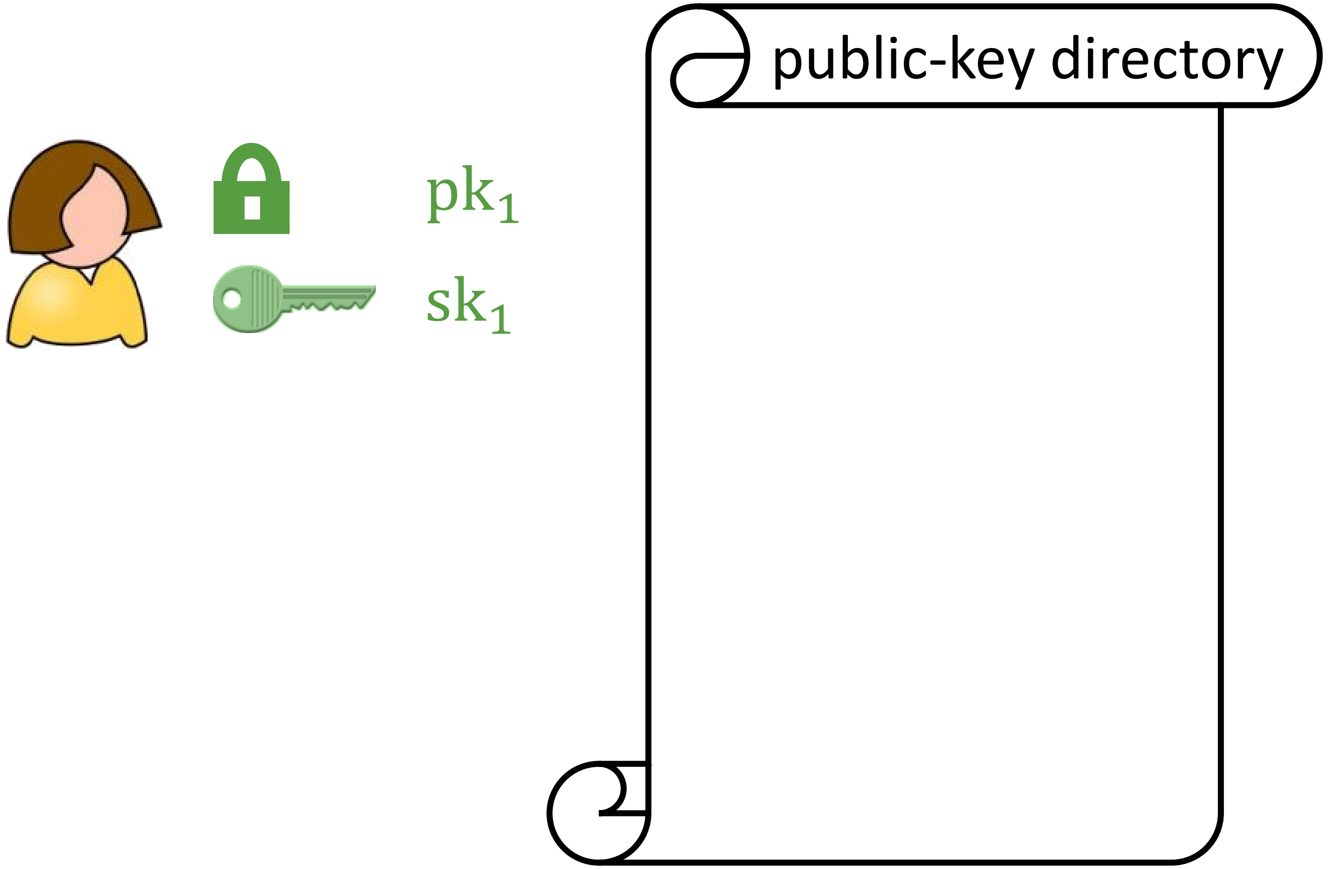


master secret key

What if the key issuer is compromised?



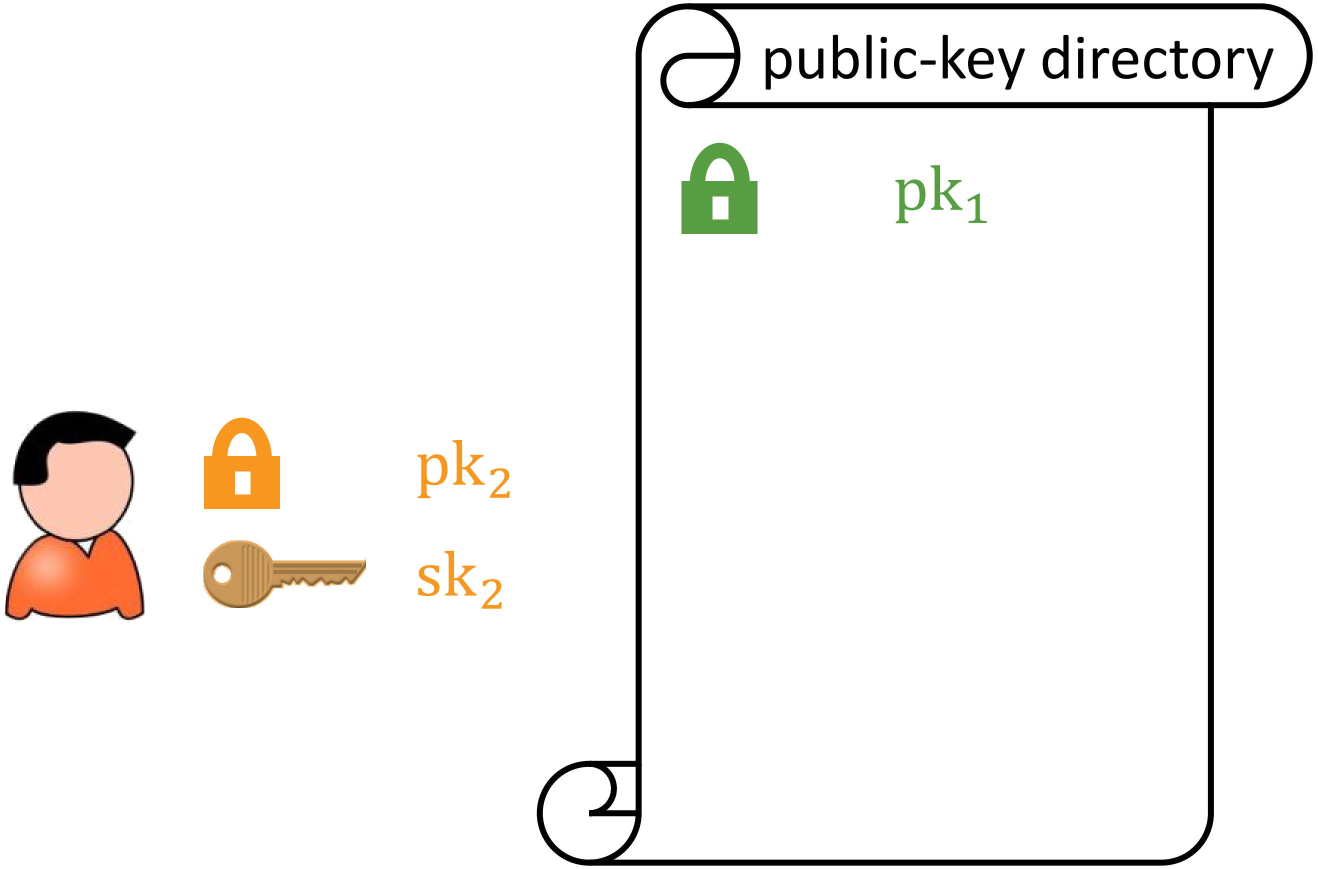
Flexible Broadcast Encryption



Users generate public/private keys independently (as in public-key encryption)

Broadcast encryption without a central authority

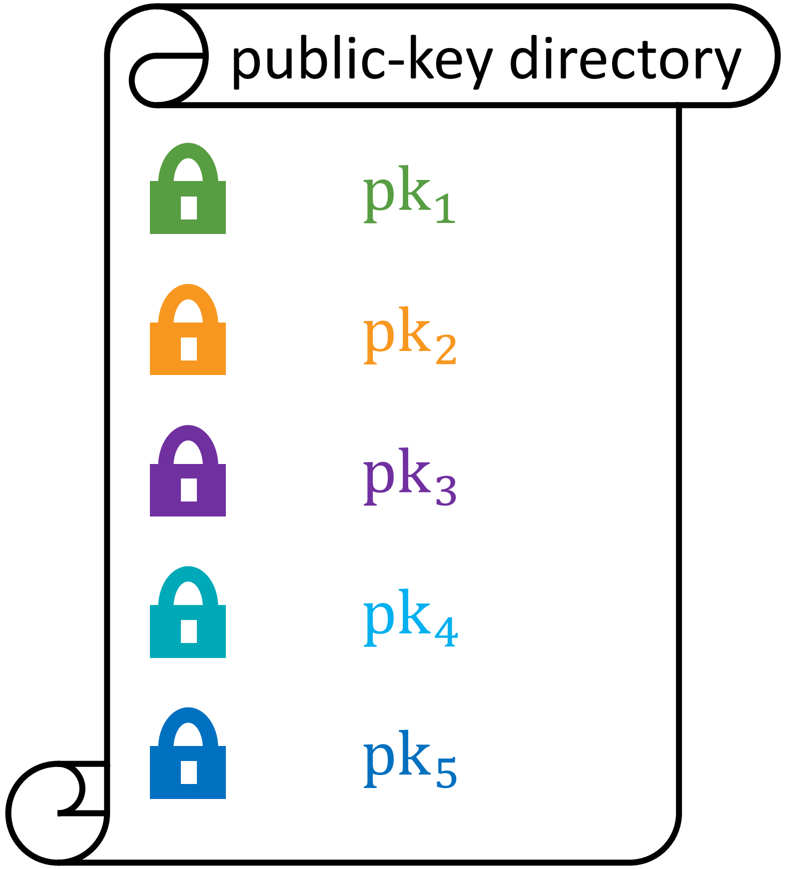
Flexible Broadcast Encryption



Users generate public/private keys independently (as in public-key encryption)

Broadcast encryption without a central authority

Flexible Broadcast Encryption



public parameters

$\text{Encrypt}(\text{pp}, \{\text{pk}_i\}_{i \in S}, m) \rightarrow \text{ct}$

Can encrypt a message m to any set of public keys

Efficiency: $|\text{ct}| = |m| + \text{poly}(\lambda, \log|S|)$

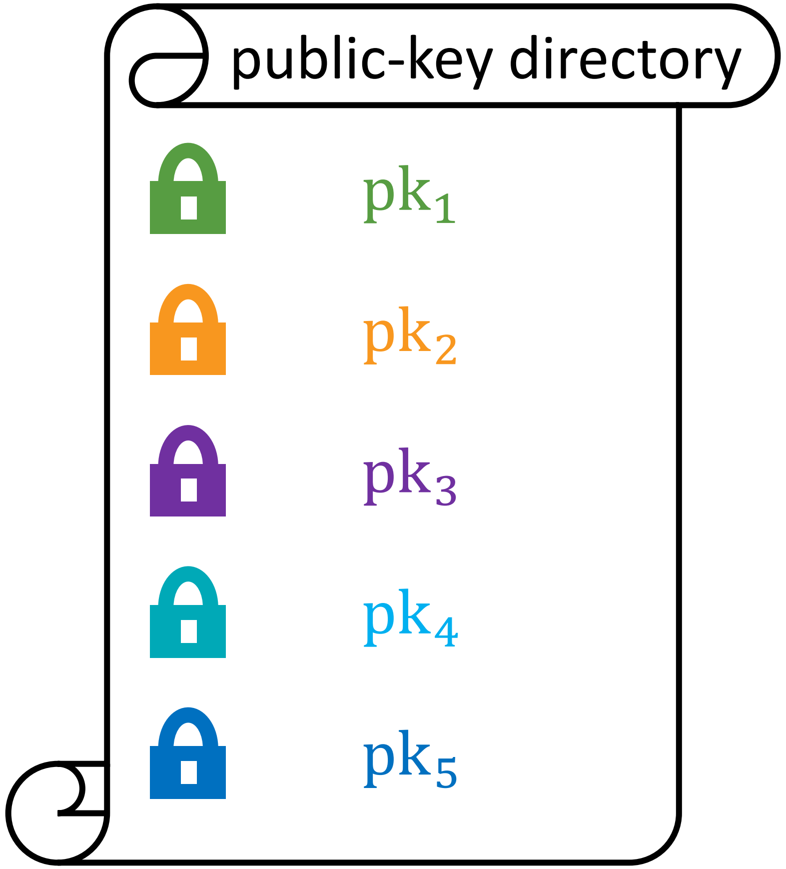
$\text{Decrypt}(\text{pp}, \{\text{pk}_i\}_{i \in S}, \text{sk}, \text{ct}) \rightarrow m$

Any secret key associated with broadcast set can decrypt

Decryption does requires knowledge of public keys in broadcast set

Broadcast encryption without a central authority

Flexible Broadcast Encryption



$\text{Encrypt}(\text{pp}, \{\text{pk}_i\}_{i \in S}, m) \rightarrow \text{ct}$

$\text{Decrypt}(\text{pp}, \{\text{pk}_i\}_{i \in S}, \text{sk}, \text{ct}) \rightarrow m$

Security: Users outside the set learn nothing about message (even if they collude)

Generalizes notion of *distributed broadcast encryption* from [BZ14]

Distributed broadcast encryption: keys are generated for a particular index, can encrypt to a set of keys occupying different indices

Broadcast encryption without a central authority

More Broadly: *Trustless* Cryptography

Functional encryption [BSW11,O'N10]**:** augment public-key encryption with fine-grained decryption capabilities

Limitation: secret keys are issued by a **central trusted authority**

Recently: removing trust from functional encryption

identity-based encryption → registration-based encryption

[GHMR18, GHMRS19, GV20]

attribute-based encryption → registered attribute-based encryption

[HLWW23]

broadcast encryption → distributed/flexible broadcast encryption

[BZ14]

functional encryption → registered functional encryption

[FFMMRV23, DP23]

More Broadly: *Trustless* Cryptography

Removing trust for functionalities beyond identity-based encryption often requires stronger cryptographic machinery

Recently: removing trust from functional encryption

identity-based encryption → registration-based encryption

[GHMR18, GHMRS19, GV20]

attribute-based encryption → registered attribute-based encryption

[HLWW23]

Registered attribute-based encryption:

- Pairing-based construction [HLWW23]: bounded number of users, large CRS, Boolean formula policies
- Indistinguishability obfuscation [HLWW23]: unbounded users, transparent setup, arbitrary policies

More Broadly: *Trustless* Cryptography

Removing trust for functionalities beyond identity-based encryption often requires stronger cryptographic machinery

Recently: removing trust from functional encryption

Distributed broadcast and registered functional encryption only known from indistinguishability obfuscation (iO)

broadcast encryption → distributed/flexible broadcast encryption

[BZ14]

functional encryption → registered functional encryption

[FFMMRV23, DP23]

This Work

Can we build trustless encryption schemes from weaker tools than indistinguishability obfuscation?

Our focus: plain witness encryption

- Witness encryption seemingly easier to realize than indistinguishability obfuscation [BJKPW18, CVW18, Tsa22, VWW22]
- Does not imply iO in a black-box manner [GMM18]

Witness encryption commonly regarded as "obfustopia" primitive and yet seems much weaker than iO

This work: new tools for realizing obfustopia primitives from plain witness encryption

Our Results

Can we build trustless encryption schemes from weaker tools than indistinguishability obfuscation?

Using witness encryption (and LWE), we obtain:

Flexible broadcast encryption

- **Previously:** distributed broadcast encryption from iO [BZ14]

Registered ABE for general policies and unbounded number of users

- **Previously:** only known from iO [HLWW23]

Optimal broadcast encryption (centralized)

- **Previously:** broadcast encryption not previously known from plain witness encryption (but known from iO, evasive LWE, or pairings + lattices)

New technique: function-binding hash functions

Witness Encryption

[GGSW13]

Defined with respect to an NP relation $\mathcal{R}$
(and associated NP language $\mathcal{L}$)

$\text{Encrypt}(x, m) \to \text{ct}$

Encrypts a message m with respect to a statement x

$\text{Decrypt}(w, \text{ct}) \to m$

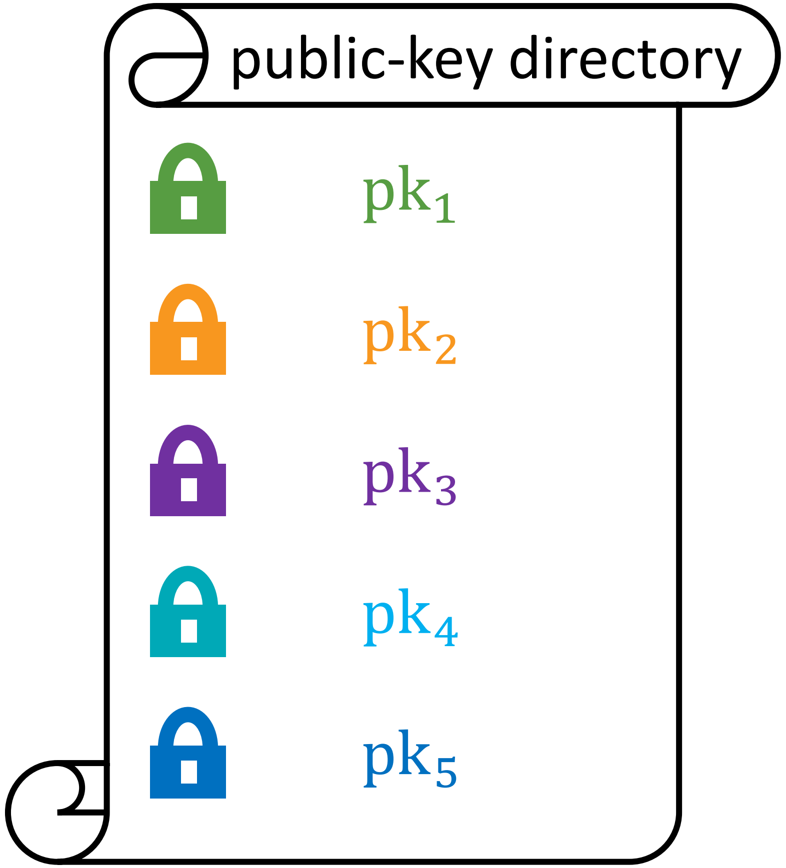
Decrypts a ciphertext given knowledge of an associated NP witness w

Functionality: if $\mathcal{R}(x, w) = 1$, decryption recovers the message m

Security: if $x \notin \mathcal{L}$, then ct hides message

Building Flexible Broadcast Encryption

Consider an approach using indistinguishability obfuscation:



Public parameters: pk for a (vanilla) public-key encryption scheme

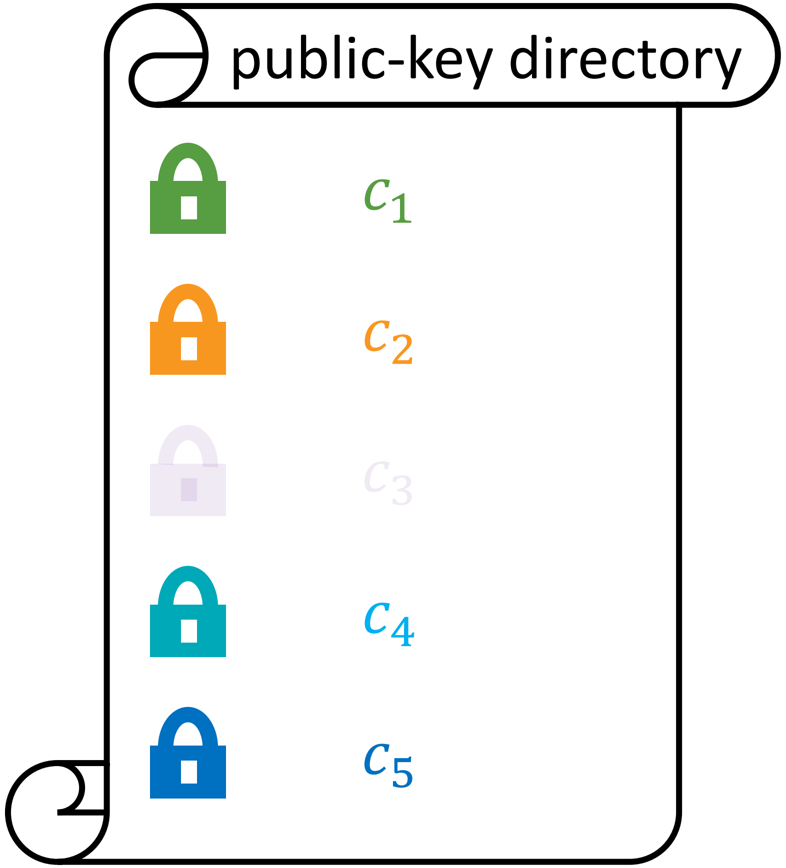
User public key: encryption of 1 with randomness r:

$$c \leftarrow \mathrm{Encrypt}(\mathrm{pk}, 1 ; r)$$

User secret key: randomness r

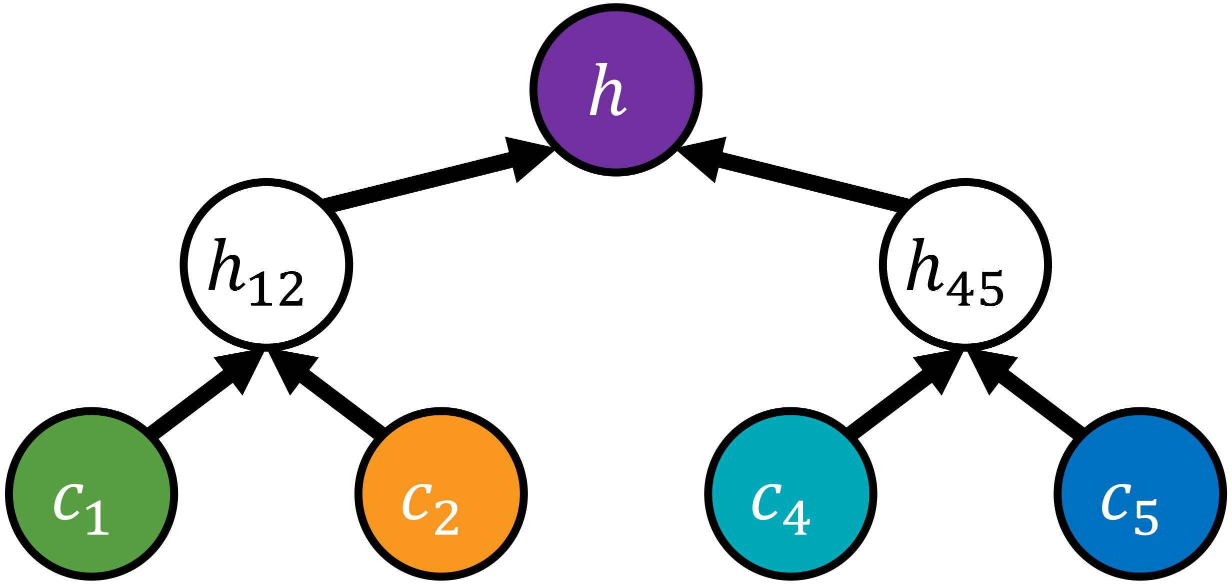
Building Flexible Broadcast Encryption

Consider an approach using indistinguishability obfuscation:



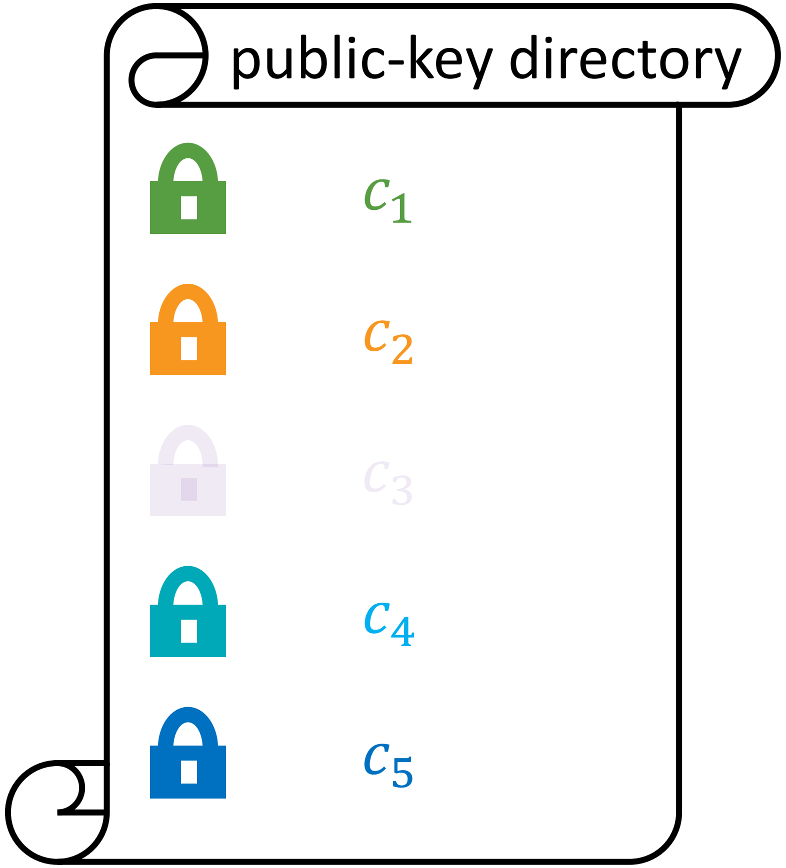
Encrypt to $S = \{c_1, c_2, c_4, c_5\}$

Step 1: Construct Merkle tree on S



Building Flexible Broadcast Encryption

Consider an approach using indistinguishability obfuscation:



Encrypt to $S = \{c_1, c_2, c_4, c_5\}$

Step 2: Obfuscate the following program

On input $\big((i, c_i), \pi_i, r_i\big)$:

- **Membership in $\boldsymbol{S}$:** Check that π_i is a Merkle inclusion proof for key c_i at position i with respect to the hash h
- **Knowledge of secret key:** Check that r_i is the secret key:

$$c_i = \mathrm{Encrypt}(\mathrm{pk}, 1; r_i)$$

If both checks pass, output m. Otherwise, output $\perp$.

Hard-coded: public parameter pk, hash S, message m

Proof Strategy

Encrypt to $S = \{c_1, c_2, c_4, c_5\}$

On input (i, c_i, π_i, r_i):

- **Membership in S:** Check that π_i is valid for c_i at position i with respect to h
- **Knowledge of secret key:** Check that $r_i = \text{Encrypt}(\text{pk}, 1; r_i)$

Output m if checks pass and $\perp$ otherwise.

Step 1: Replace $c_i \leftarrow \text{Encrypt}(\text{pk}, 0)$

Indistinguishable by semantic security of public-key encryption scheme

Proof Strategy

Encrypt to $S = \{c_1, c_2, c_4, c_5\}$

On input (i, c_i, π_i, r_i):

- **Membership in S:** Check that π_i is valid for c_i at position i with respect to h
- **Knowledge of secret key:** Check that $r_i = \text{Encrypt}(\text{pk}, 1; r_i)$

Output m if checks pass and $\perp$ otherwise.

This condition is unsatisfiable for $c_i \in S$

Problem: But could still exist valid openings for $c_i' \notin S$

Step 1: Replace $c_i \leftarrow \text{Encrypt}(\text{pk}, 0)$

Indistinguishable by semantic security of public-key encryption scheme

Proof Strategy

Encrypt to $S = \{c_1, c_2, c_4, c_5\}$

On input (i, c_i, π_i, r_i):

- **Membership in S:** Check that π_i is valid for c_i at position i with respect to h
- **Knowledge of secret key:** Check that $r_i = \text{Encrypt}(\text{pk}, 1; r_i)$

Output m if checks pass and $\perp$ otherwise.

Ensures that the only opening at index $i = 1$ to h is $c_1 = \text{Encrypt}(\text{pk}, 0)$

Step 2: Use a somewhere statistically-binding (SSB) hash function to compute h and statistically bind at index $i = 1$

Implication: On all inputs where $i = 1$, program will output $\perp$

Proof Strategy

Encrypt to $S = \{c_1, c_2, c_4, c_5\}$

Identical functionality so indistinguishable under iO

On input (i, c_i, π_i, r_i):

- **Membership in $\boldsymbol{S}$:** Check that π_i is valid for c_i at position i with respect to h
- **Knowledge of secret key:** Check that $r_i = \text{Encrypt}(\text{pk}, 1; r_i)$

Output m if checks pass and $\perp$ otherwise.

$\approx$

On input (i, c_i, π_i, r_i):

- **Index threshold:** $i > 1$
- **Membership in $\boldsymbol{S}$:** Check that π_i is valid for c_i at position i with respect to h
- **Knowledge of secret key:** Check that $r_i = \text{Encrypt}(\text{pk}, 1; r_i)$

Output m if checks pass $\perp$ otherwise.

Step 2: Use a somewhere statistically-binding (SSB) hash function to compute h and statistically bind at index $i = 1$

Implication: On all inputs where $i = 1$, program will output $\perp$

Proof Strategy

Encrypt to $S = \{c_1, c_2, c_4, c_5\}$

Identical functionality so indistinguishable under iO

On input (i, c_i, π_i, r_i):

- **Index threshold:** $i > 1$
- **Membership in S:** Check that π_i is valid for c_i at position i with respect to h
- **Knowledge of secret key:** Check that $r_i = \text{Encrypt}(\text{pk}, 1; r_i)$

Output m if checks pass $\perp$ otherwise.

$\approx$

On input (i, c_i, π_i, r_i):

- **Index threshold:** $i > 2$
- **Membership in S:** Check that π_i is valid for c_i at position i with respect to h
- **Knowledge of secret key:** Check that $r_i = \text{Encrypt}(\text{pk}, 1; r_i)$

Output m if checks pass $\perp$ otherwise.

Step 3: Use a somewhere statistically-binding (SSB) hash function to compute h and statistically bind at index $i = 2$

Implication: On all inputs where $i = 2$, program will output $\perp$

Proof Strategy

Encrypt to $S = \{c_1, c_2, c_4, c_5\}$

Identical functionality so indistinguishable under iO

On input (i, c_i, π_i, r_i): • **Index threshold:** $i > 4$ • **Membership in S:** Check that π_i is valid for c_i at position i with respect to h • **Knowledge of secret key:** Check that $r_i = \text{Encrypt}(\text{pk}, 1; r_i)$ Output m if checks pass $\perp$ otherwise.	$\approx$	On input (i, c_i, π_i, r_i): Output $\perp$.

Repeat process for each public-key in S

Final program requires that input threshold $i > |S|$, which is **never** satisfied

Ciphertext indistinguishable from program that outputs $\perp$ on all inputs

Replacing iO with Witness Encryption

Encrypt to $S = \{c_1, c_2, c_4, c_5\}$

On input (i, c_i, π_i, r_i):

- **Membership in S:** Check that π_i is valid for c_i at position i with respect to h
- **Knowledge of secret key:** Check that $r_i = \text{Encrypt}(\text{pk}, 1; r_i)$

Output m if checks pass and $\perp$ otherwise.

This program is checking an NP relation!

Statement: (pk, h)

Witness: (i, c_i, π_i, r_i)

What happens if we replace iO with witness encryption?

Challenge: need to argue that there are no witnesses for (pk, h)

Can replace all $c_i \in S$ with encryptions of 0, but there can still be openings to h that are encryptions of 1 (since h is computationally binding)

Replacing iO with Witness Encryption

Encrypt to $S = \{c_1, c_2, c_4, c_5\}$

On input (i, c_i, π_i, r_i):

- **Membership in S:** Check that π_i is valid for c_i at position i with respect to h
- **Knowledge of secret key:** Check that $r_i = \text{Encrypt}(\text{pk}, 1; r_i)$

Output m if checks pass and $\perp$ otherwise.

With obfuscation: h can statistically bind to one index; obfuscated program saves progress →

What happens if we replace iO with witness encryption?

Challenge: need to argue that there are no witnesses for (pk, h)

Can replace all $c_i \in S$ with encryptions of 0, but there can still be openings to h that are encryptions of 1 (since h is computationally binding)

No analog with plain witness encryption

Replacing iO with Witness Encryption

Encrypt to $S = \{c_1, c_2, c_4, c_5\}$

On input (i, c_i, π_i, r_i):

- **Membership in S:** Check that π_i is valid for c_i at position i with respect to h

Statistically binding to all indices results in long ciphertext (linear in $|S|$)

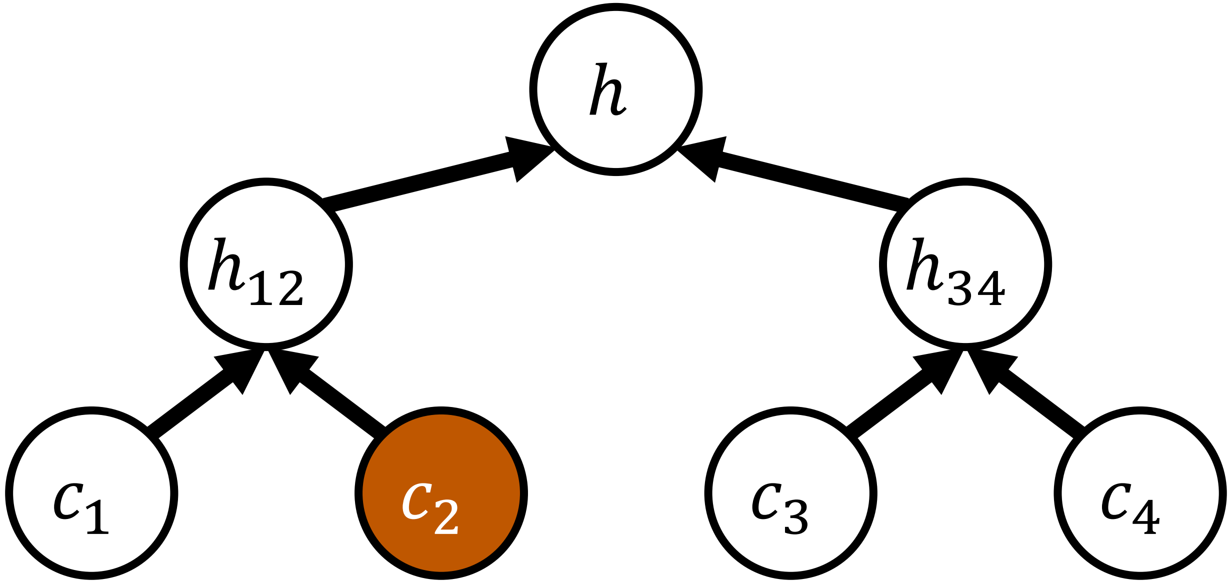
With obfuscation: h can statistically bind to one index; obfuscated program saves progress → No analog with plain witness encryption

What happens if we replace iO with witness encryption?

Challenge: need to argue that there are no witnesses for (pk, h)

Can replace all $c_i \in S$ with encryptions of 0, but there can still be openings to h that are encryptions of 1 (since h is computationally binding)

Function-Binding Hash Functions

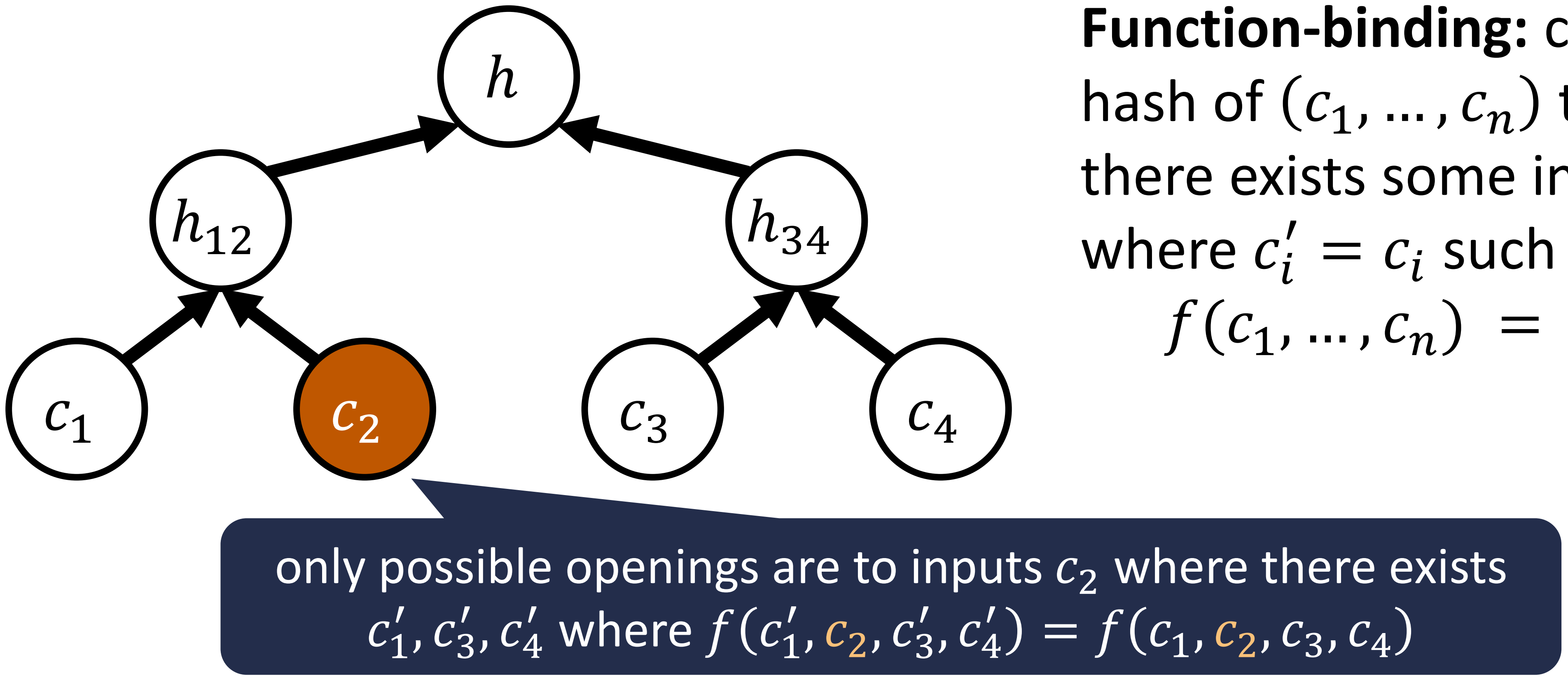


Somewhere statistically binding: can only open hash of $(c_1, \ldots, c_n)$ to value c_i at a particular index i

if hash key is binding at $i = 2$, then c_2 is only possible opening for h at index 2

Function-Binding Hash Functions

Our approach: hash function h statistically binds to a function of the input



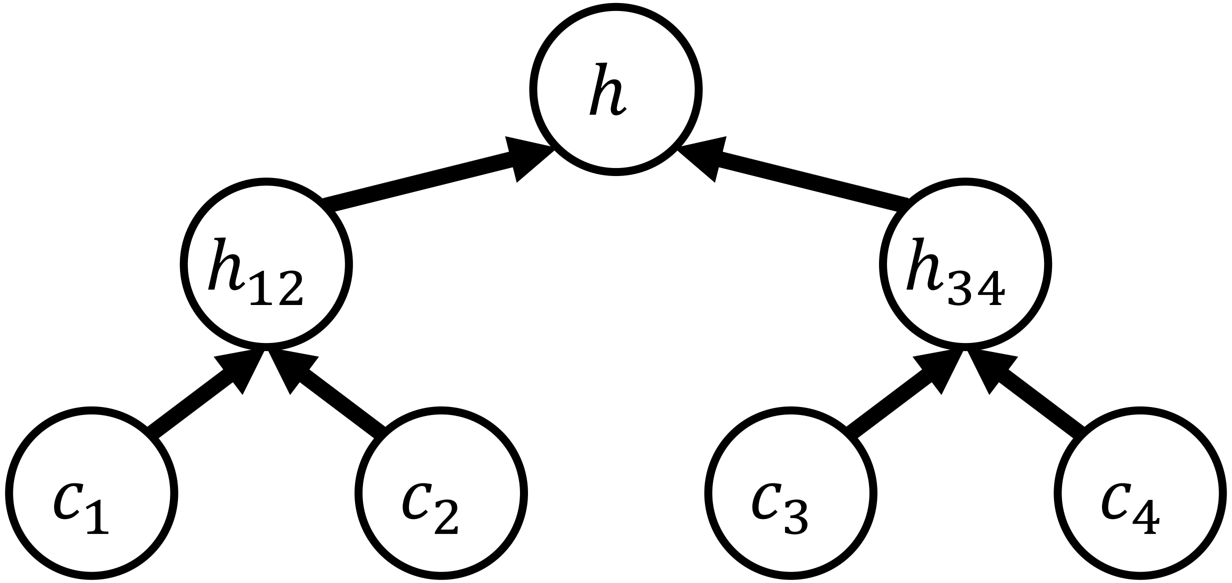
Function-binding: can only open hash of $(c_1, \ldots, c_n)$ to a value c_i if there exists some input $(c'_1, \ldots, c'_n)$ where $c'_i = c_i$ such that

$$f(c_1, \ldots, c_n) = f(c'_1, \ldots, c'_n)$$

Hash key associated with the specific function f

Function-Binding Hash Functions

This work: function-binding for disjunction-of-predicates class



$$f(c_1, \ldots, c_n) = \bigvee_{i \in [n]} g(c_i)$$

$g(c_i) \in \{0,1\}$ is an arbitrary predicate

Suppose $g(c_i) = 0$ for all $i \in [n]$

Then $f(c_1, \ldots, c_n) = 0$

Guarantee: Does not exist *any* openings for h to an input c_i where $g(c_i) = 1$

Function-binding hash functions bind to a **global** property of the input

Using Function-Binding Hash Functions

Encrypt to $S = \{c_1, c_2, c_4, c_5\}$

On input (i, c_i, π_i, r_i):

- **Membership in S:** Check that π_i is valid for c_i at position i with respect to h
- **Correct decryption key:** Check that $r_i = \mathrm{Encrypt}(\mathrm{pk}, 1; r_i)$

Output m if checks pass and $\perp$ otherwise.

This program is checking an NP relation!

Statement: (pk, h)

Witness: (i, c_i, π_i, r_i)

Use function-binding hash function for the function

$$f_{\mathrm{sk}}(c_1, \ldots, c_n) = \bigvee_{i \in [n]} \mathrm{Decrypt}(\mathrm{sk}, c_i)$$

Recall: $c_i = \mathrm{Encrypt}(\mathrm{pk}, 1)$

Note: Will require that the hash key **hides** the function (analogous to index hiding in SSB)

Using Function-Binding Hash Functions

Encrypt to $S = \{c_1, c_2, c_4, c_5\}$

On input (i, c_i, π_i, r_i):

- **Membership in S:** Check that π_i is valid for c_i at position i with respect to h
- **Correct decryption key:** Check that $r_i = \text{Encrypt}(\text{pk}, 1; r_i)$

Output m if checks pass and $\perp$ otherwise.

This program is checking an NP relation!

Statement: (pk, h)

Witness: (i, c_i, π_i, r_i)

Security proof:

- **Step 1:** Switch each c_i in challenge ciphertext to encryptions of 0 (as before)
- **Step 2:** Switch hash function to function bind on f_{sk}

$$f_{\text{sk}}(c_1, \dots, c_n) = \bigvee_{i \in [n]} \text{Decrypt}(\text{sk}, c_i)$$

Using Function-Binding Hash Functions

Encrypt to $S = \{c_1, c_2, c_4, c_5\}$

On input (i, c_i, π_i, r_i):

- **Membership in S:** Check that π_i is valid for c_i at position i with respect to h
- **Correct decryption key:** Check that $r_i = \text{Encrypt}(\text{pk}, 1; r_i)$

Output m if checks pass and $\perp$ otherwise.

This program is checking an NP relation!

Statement: (pk, h)
Witness: (i, c_i, π_i, r_i)

Security proof:

- **Step 1:** Switch each c_i in challenge ciphertext to encryptions of 0 (as before)
- **Step 2:** Switch hash function to function bind on f_{sk}

For challenge ciphertext:
$f_{\text{sk}}(c_1, \ldots, c_n) = 0$

Function binding: no openings exist for any c_i that is an encryption of 1

Using Function-Binding Hash Functions

Encrypt to $S = \{c_1, c_2, c_4, c_5\}$

On input (i, c_i, π_i, r_i):

- **Membership in S:** Check that π_i is valid for c_i at position i with respect to h
- **Correct decryption key:** Check that $r_i = \text{Encrypt}(\text{pk}, 1; r_i)$

Output m if checks pass and $\perp$ otherwise.

This program is checking an NP relation!

Statement: (pk, h)

Witness: (i, c_i, π_i, r_i)

Security proof:

- **Step 1:** Switch each c_i in challenge ciphertext to encryptions of 0 (as before)
- **Step 2:** Switch hash function to function bind on f_{sk}
- **Step 3:** No valid witness exists so can appeal to security of witness encryption

$$f_{\text{sk}}(c_1, \dots, c_n) = \bigvee_{i \in [n]} \text{Decrypt}(\text{sk}, c_i)$$

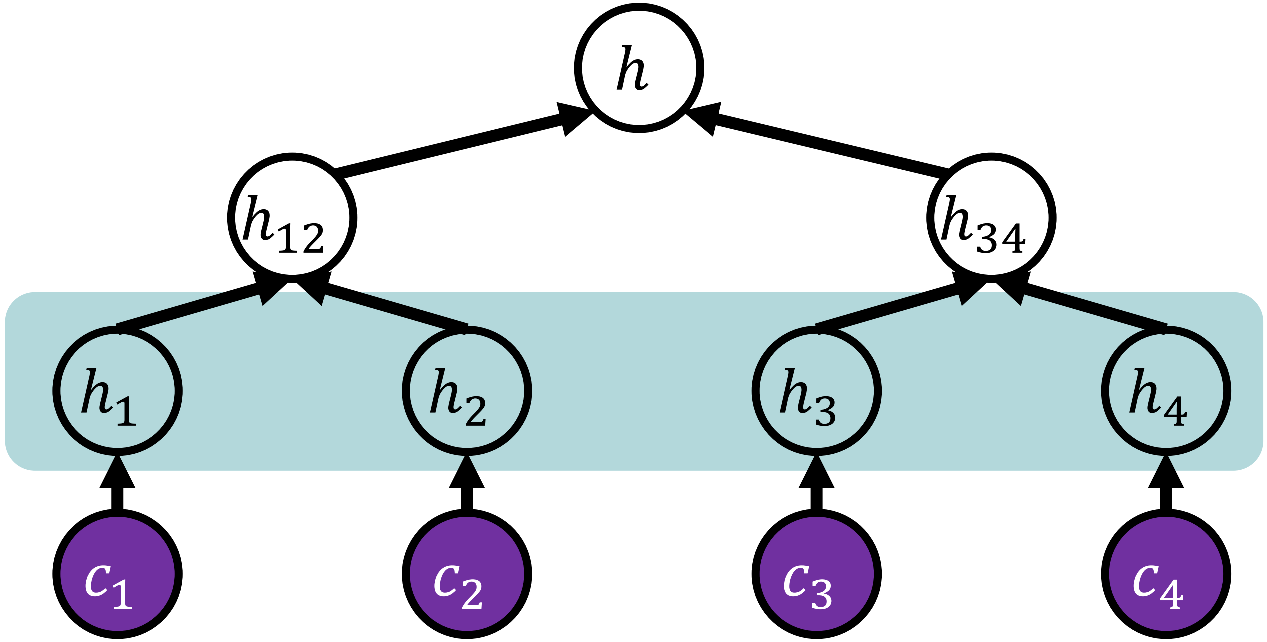
Constructing Function-Binding Hash Functions

Will focus on supporting disjunction-of-predicate class

$$f(c_1, \dots, c_n) = \bigvee_{i \in [n]} g(c_i)$$

Follows from (leveled) homomorphic encryption

(similar to constructions of SSB hash functions [HW15])



Leaf nodes: homomorphically evaluate g on input

hash key hk contains encryption of g under pk_0 (so hk hides g)

$$h_i = \text{Encrypt}(\text{pk}_0, g(c_i))$$

LHE encryption key for level zero (leaves)

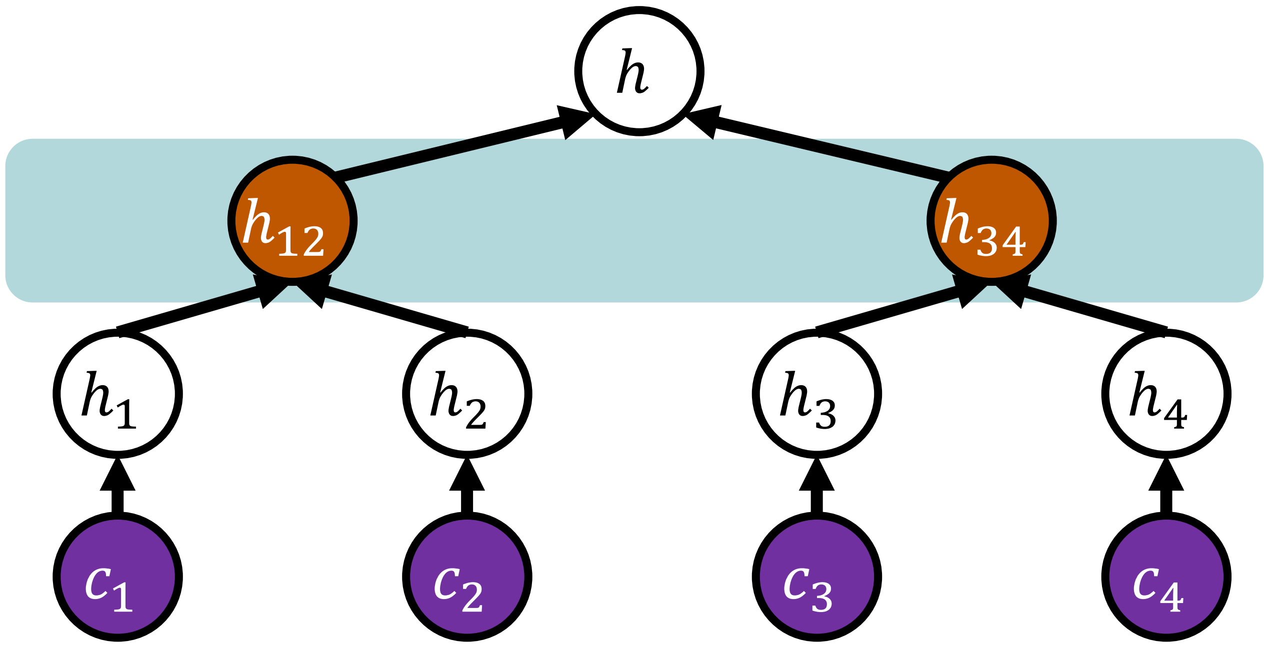
Constructing Function-Binding Hash Functions

Will focus on supporting disjunction-of-predicate class

$$f(c_1, \ldots, c_n) = \bigvee_{i \in [n]} g(c_i)$$

Follows from (leveled) homomorphic encryption

(similar to constructions of SSB hash functions [HW15])



Internal nodes: homomorphically decrypt value of child nodes and compute OR of results

$$h_{12} = \mathrm{Enc}\big(\mathrm{pk}_1, \mathrm{Dec}(\mathrm{sk}_0, h_1) \vee \mathrm{Dec}(\mathrm{sk}_1, h_2)\big)$$

pk_1: encryption key for level 1

sk_0: decryption key for level 0 (encrypted under pk_1 and part of hk)

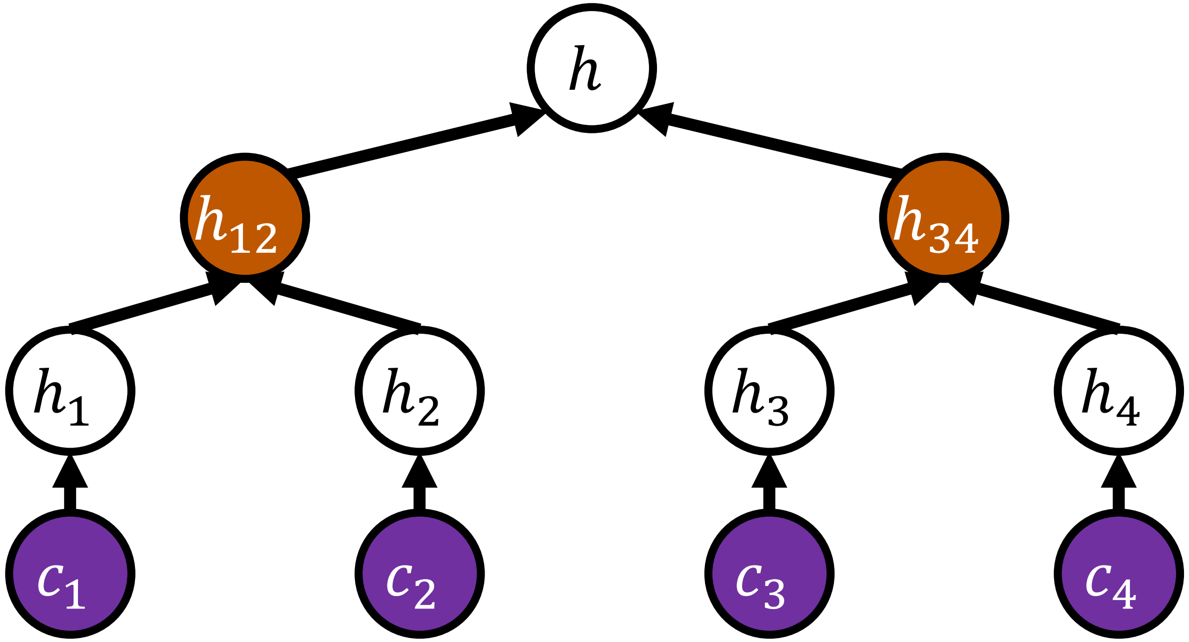
Constructing Function-Binding Hash Functions

Will focus on supporting disjunction-of-predicate class

$$f(c_1, \ldots, c_n) = \bigvee_{i \in [n]} g(c_i)$$

Follows from (leveled) homomorphic encryption

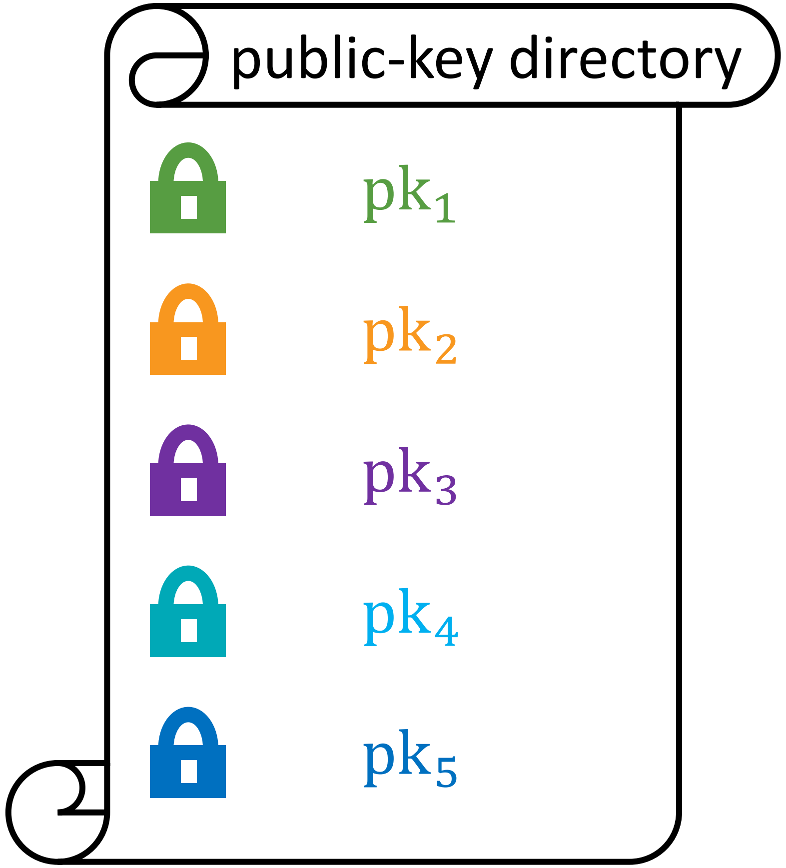
(similar to constructions of SSB hash functions [HW15])



Observe: value of root node h is an (honest) encryption of $f(c_1, \ldots, c_n)$

Function binding follows by **correctness** of the homomorphic encryption scheme (h cannot simultaneously be an encryption of 0 and 1)

Flexible Broadcast Encryption



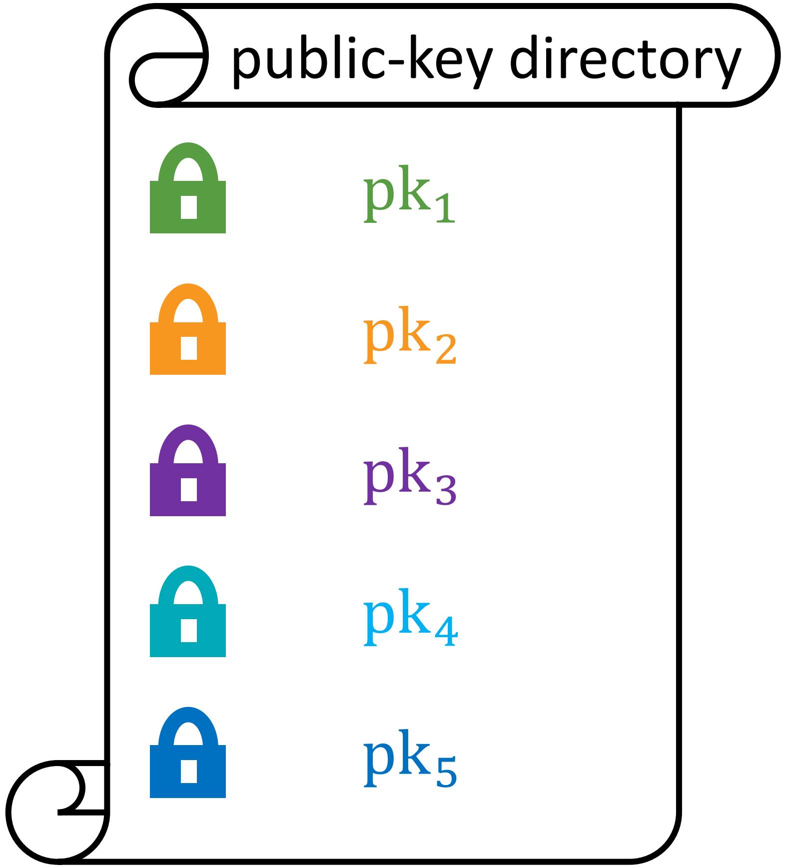
Ciphertext: Witness encryption of message with respect to hash of the public keys in the broadcast set

Decryption: "Proof of knowledge" of secret key for one of the keys in the broadcast set S

$$|\mathrm{ct}| = \mathrm{poly}(\lambda, \log|S|)$$
$$|\mathrm{sk}| = \mathrm{poly}(\lambda)$$
$$|\mathrm{pk}| = \mathrm{poly}(\lambda)$$

Does *not* yield optimal broadcast encryption in the *centralized* setting

Optimal Broadcast Encryption



Does *not* yield optimal broadcast encryption in the *centralized* setting

Approach: define the public key to be

$$\mathrm{pk}_i \leftarrow H(i)$$

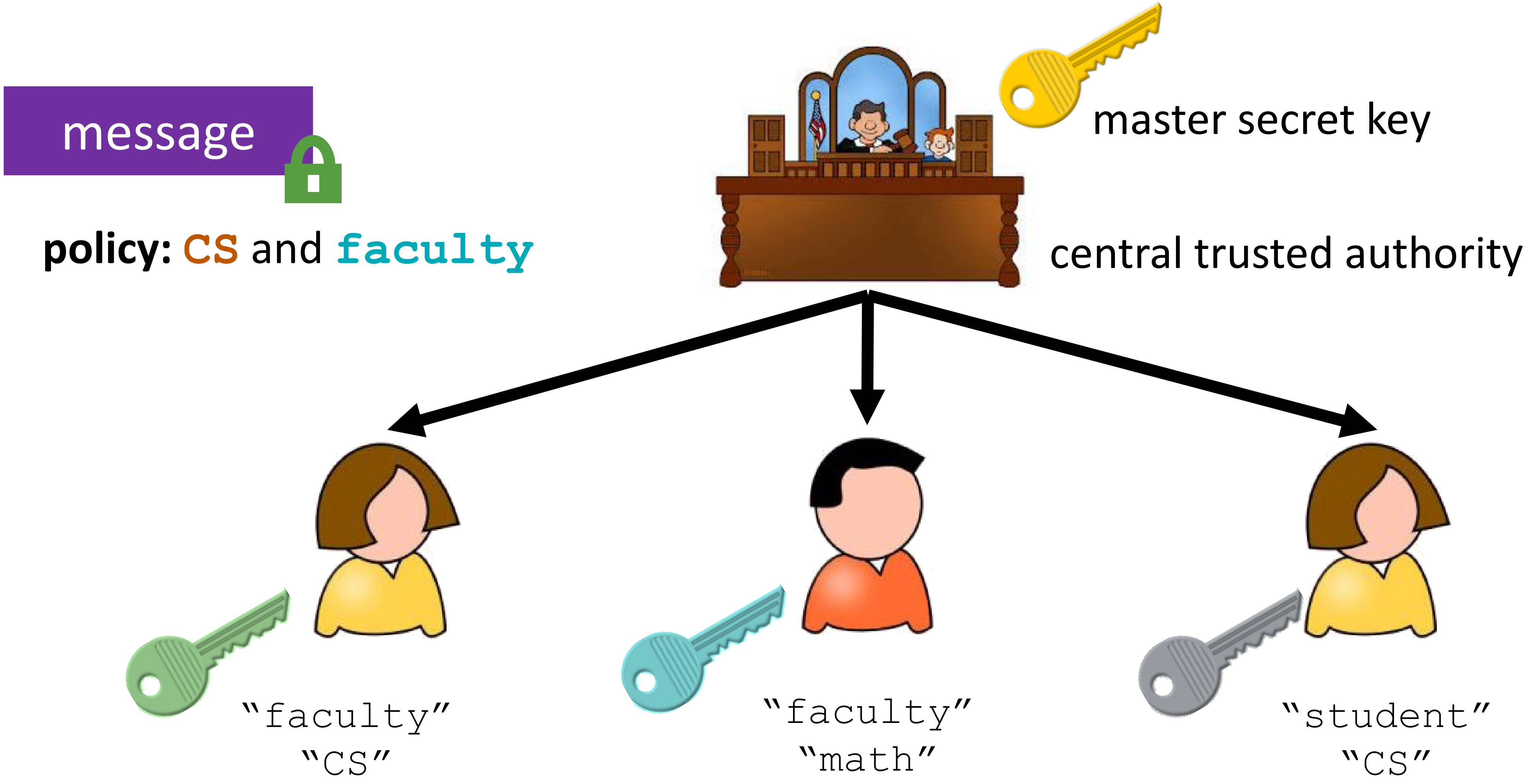
where $H(\cdot)$ is modeled as a random oracle

Use **trapdoor** to sample the secret key sk_i associated with pk_i

[see paper for details]

Ciphertext-Policy Attribute-Based Encryption

[SW05, GPSW06]

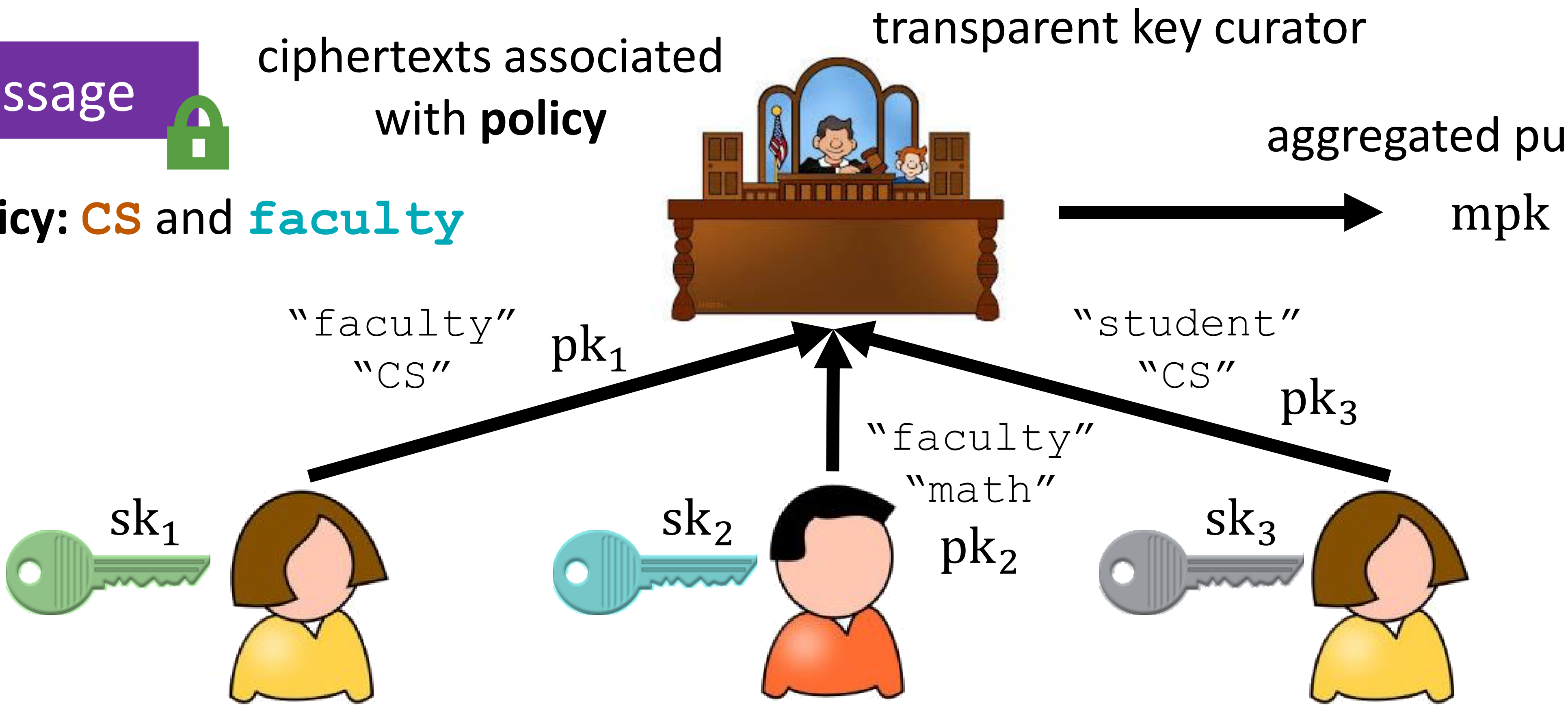


Registered Attribute-Based Encryption (ABE)

message

ciphertexts associated with **policy**

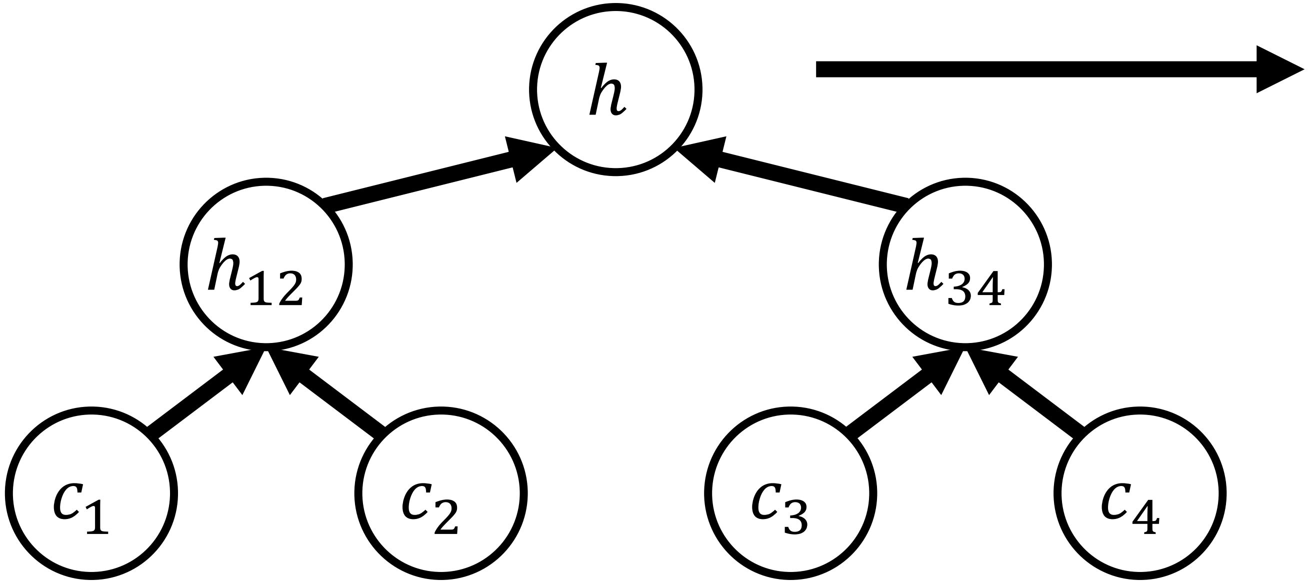
policy: CS and faculty



Users chooses their <u>own</u> public/secret key

Users join the system by registering their public key along with a set of attributes

Registered ABE from Plain Witness Encryption



aggregated master public key

$$c_i = (\mathrm{pk}_i, x_i)$$

i^{th} user's public key

i^{th} user's attribute

User registration

Encrypt to policy P

On input $(i, \mathrm{pk}_i, x_i, \pi_i, \mathrm{sk}_i)$:

- **Key is registered:** Check that π_i is valid for $c_i = (\mathrm{pk}_i, x_i)$ with respect to h
- **Knowledge of secret key:** Check that sk_i is secret key for pk_i
- **Policy satisfiability:** Check that $P(x_i) = 1$

Output m if checks pass and $\perp$ otherwise.

Proof relies on similar function-binding strategy

Summary

Can we build trustless encryption schemes from weaker tools than indistinguishability obfuscation?

This work: introduced notion of function-binding hash functions

Captures SSB hash functions as a special case

Suffices to realize new trustless cryptographic primitives from witness encryption:

Flexible broadcast encryption

Registered ABE for general policies (and unbounded number of users)

In fact, registered ABE implies flexible/distributed broadcast encryption *[see paper]*

Open Problems

New constructions of function-binding hash functions

Constructions without LWE?

Constructions for other function families?

Our FHE approach generalizes to threshold-of-predicate

Impossibility for (general) function classes?

New applications of function-binding hash functions?

(with or without witness encryption)

Thank you!