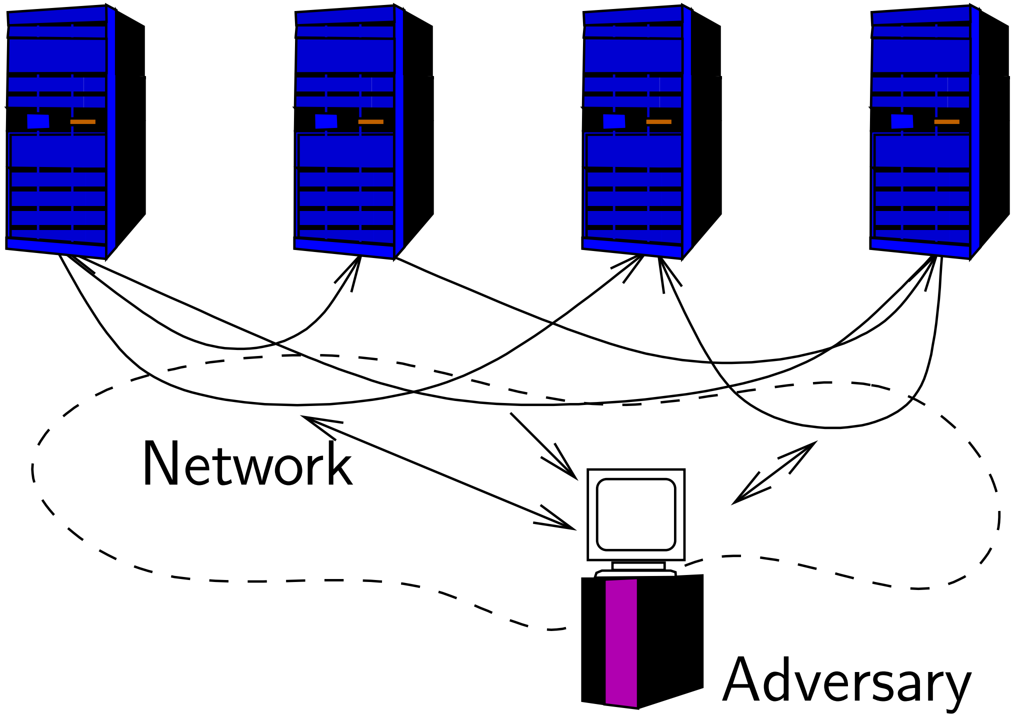
Threshold homomorphic encryption in the universally composable cryptographic library

Peeter Laud
Cybernetica AS & Tartu University
peeter.laud@ut.ee
http://www.cs.ut.ee/~peeter_l

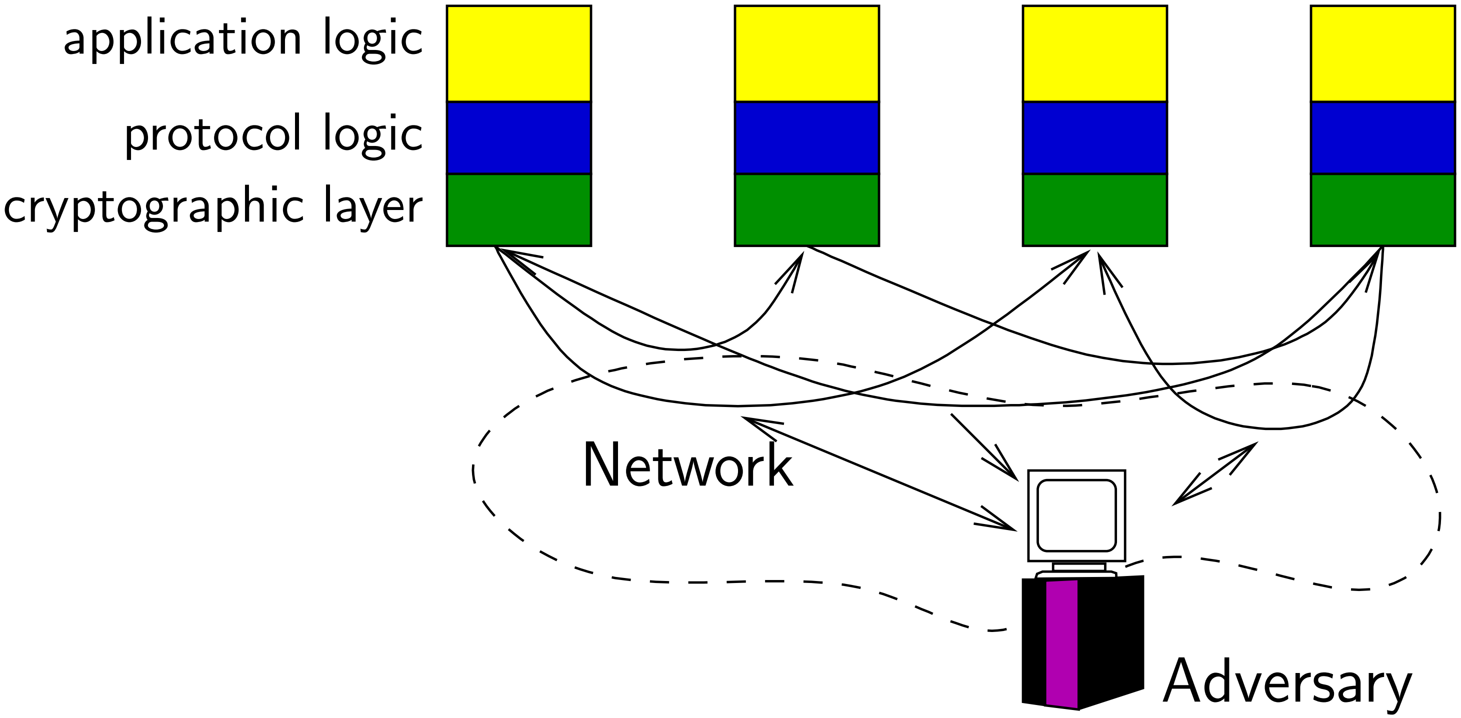
joint work with Long Ngo

A distributed system



Several sites, channels between some of them, channels may be secure, authentic or insecure.

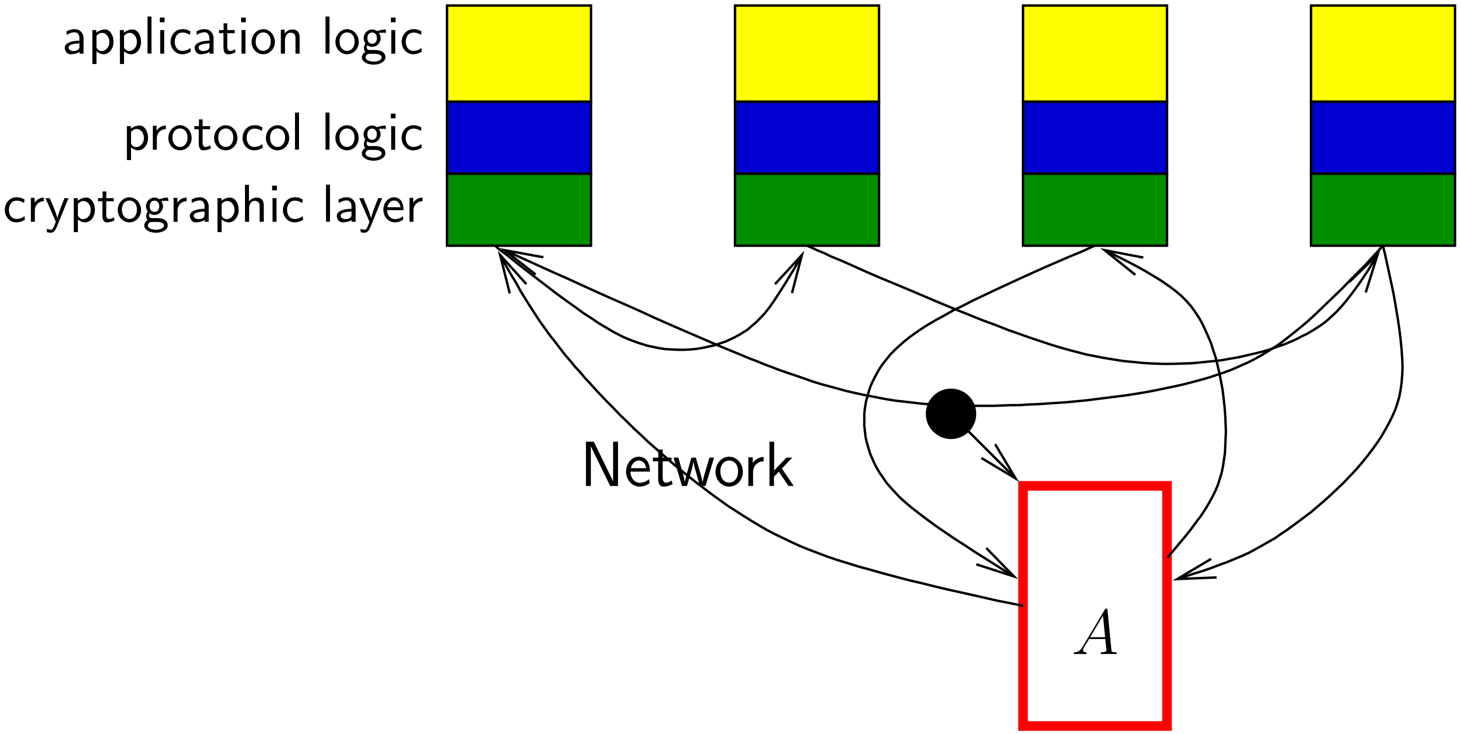
A distributed system



The instructions of a site can typically be partitioned to the following three layers.

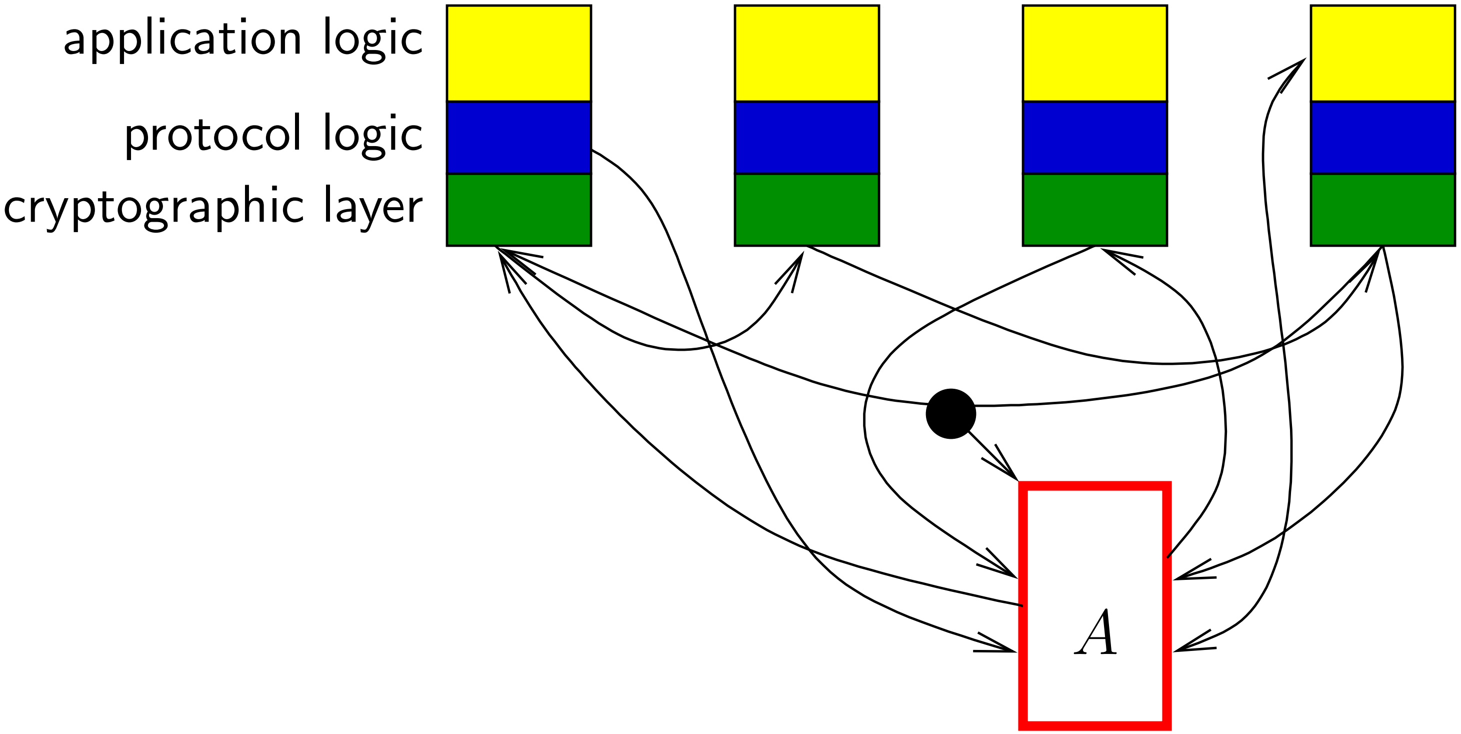
Site — a composition of three (or more) interacting Turing machines.

A distributed system



All three types of channels can be modeled with the help of secure channels.

A distributed system



The upper layers may also influence and be influenced by the adversary.

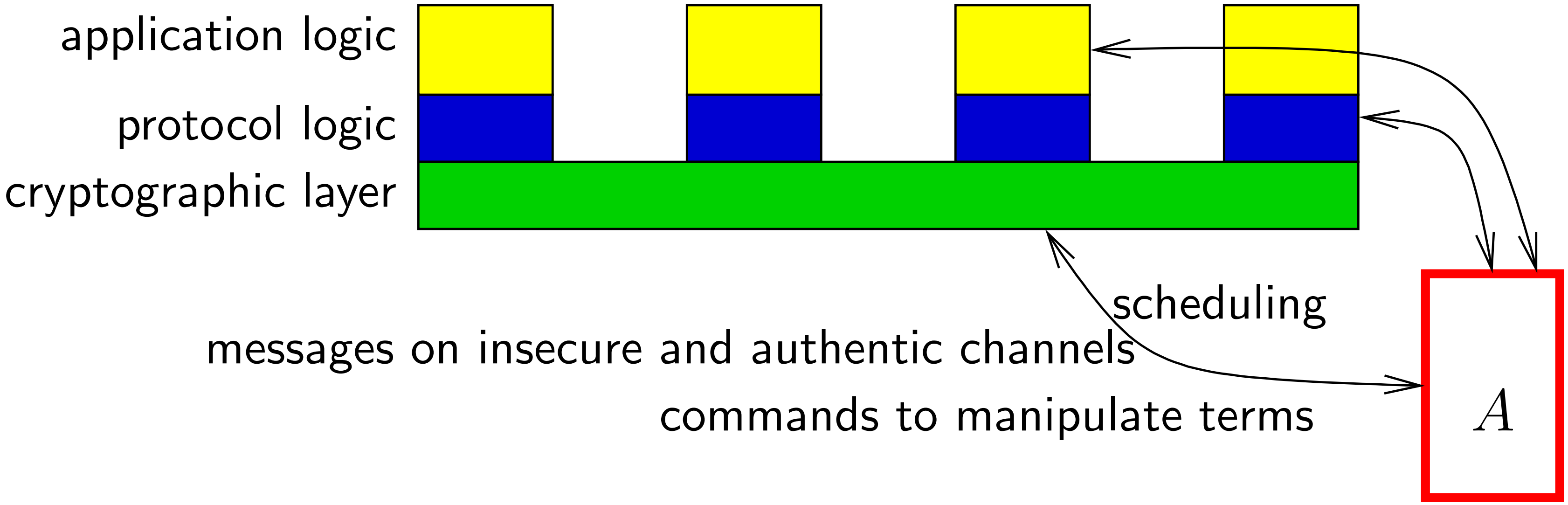
Example: I/O, timing.

The simulatable cryptographic library

- May serve as the cryptographic layer.
- Takes API calls from the layer above to
  - generate new encryption/decryption keys, encrypt and decrypt;
    - both symmetric and asymmetric encryption are present
  - generate new signature keys, sign and verify;
  - generate new MAC keys, tag and verify;
  - take and return (unstructured) data; construct and destruct tuples;
  - send messages to other parties.
- Receives messages from other parties and forwards them to the layer above.
- The overlying layer accesses all messages through handles.

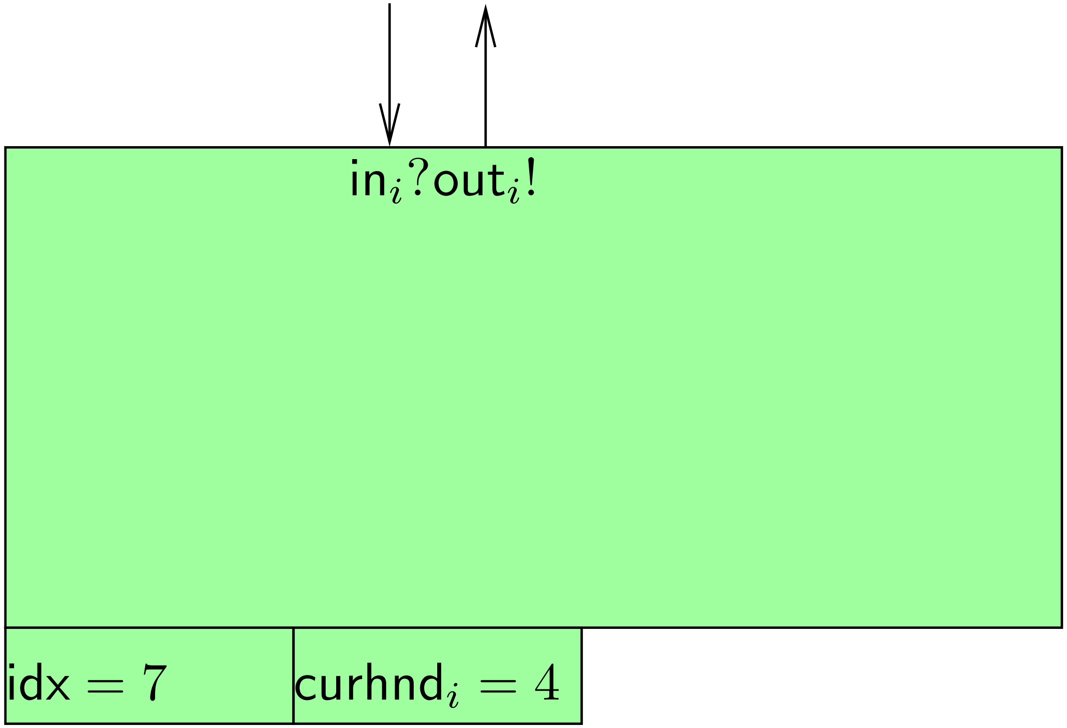
[Backes, Pfitzmann, Waidner; CCS 2003]

The abstract cryptographic library

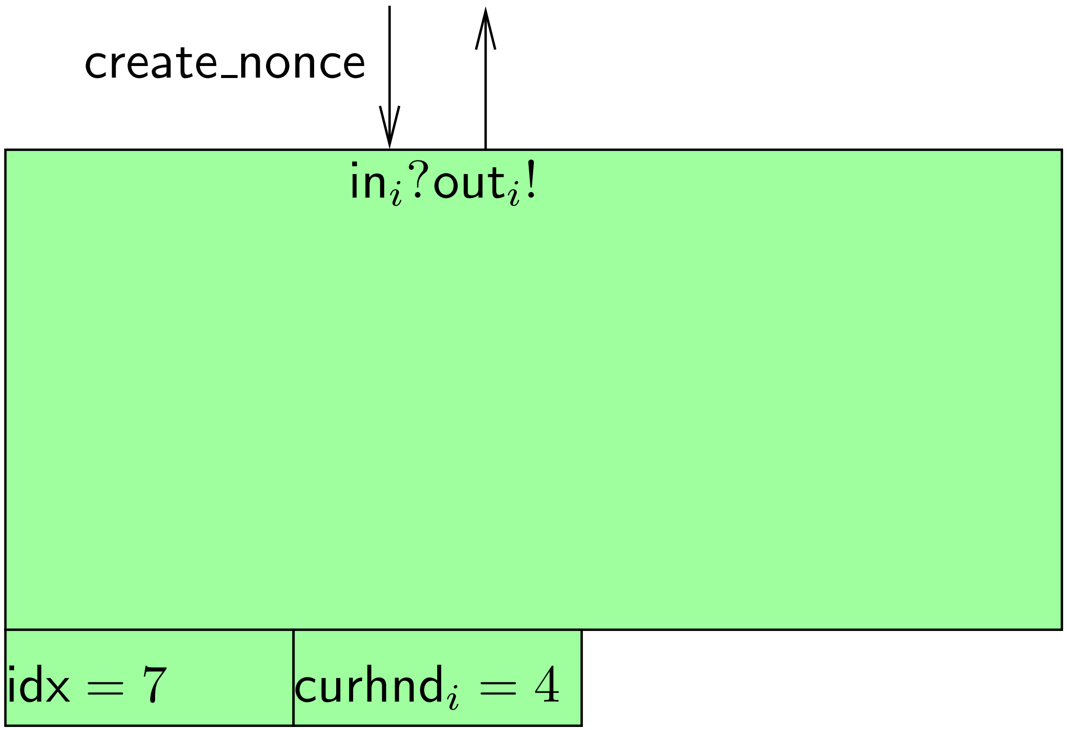

application logic
protocol logic
cryptographic layer
scheduling
messages on insecure and authentic channels
commands to manipulate terms
A

- A monolithic library — consists of a single machine.
- Cannot be directly implemented.
- Main part — a database of terms recording their structure and parties that have access to them.
- Terms in the database ≈ terms in the Dolev-Yao model.
- Possible operations also similar to the Dolev-Yao model.

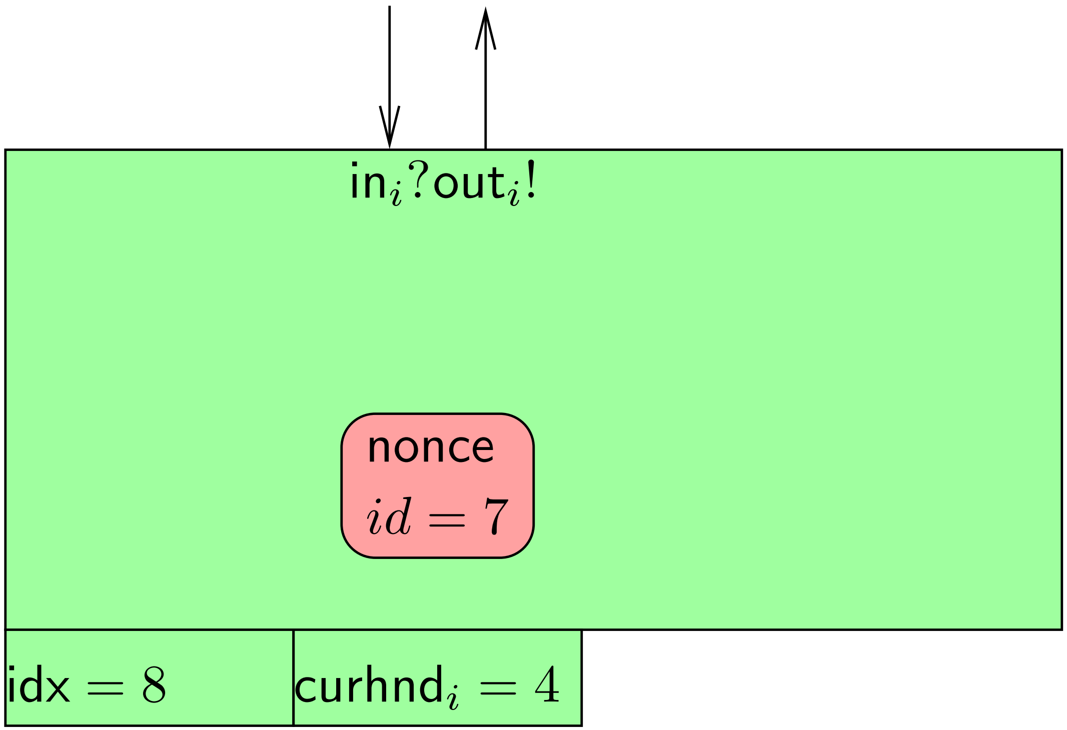
Operations: example 1



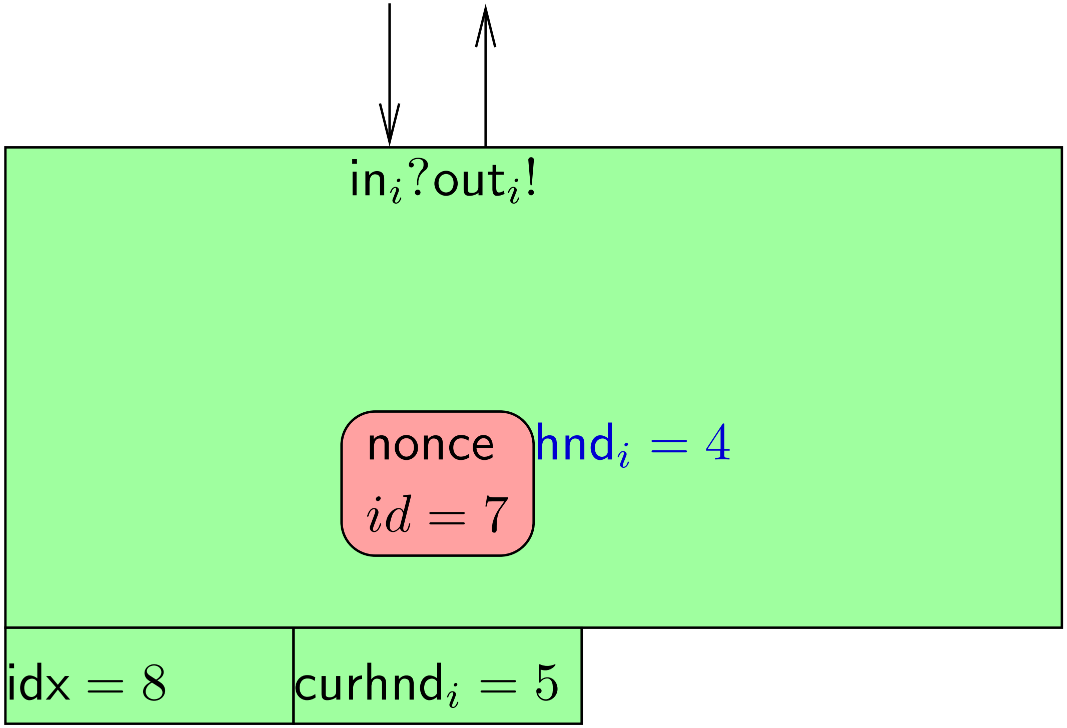
Operations: example 1



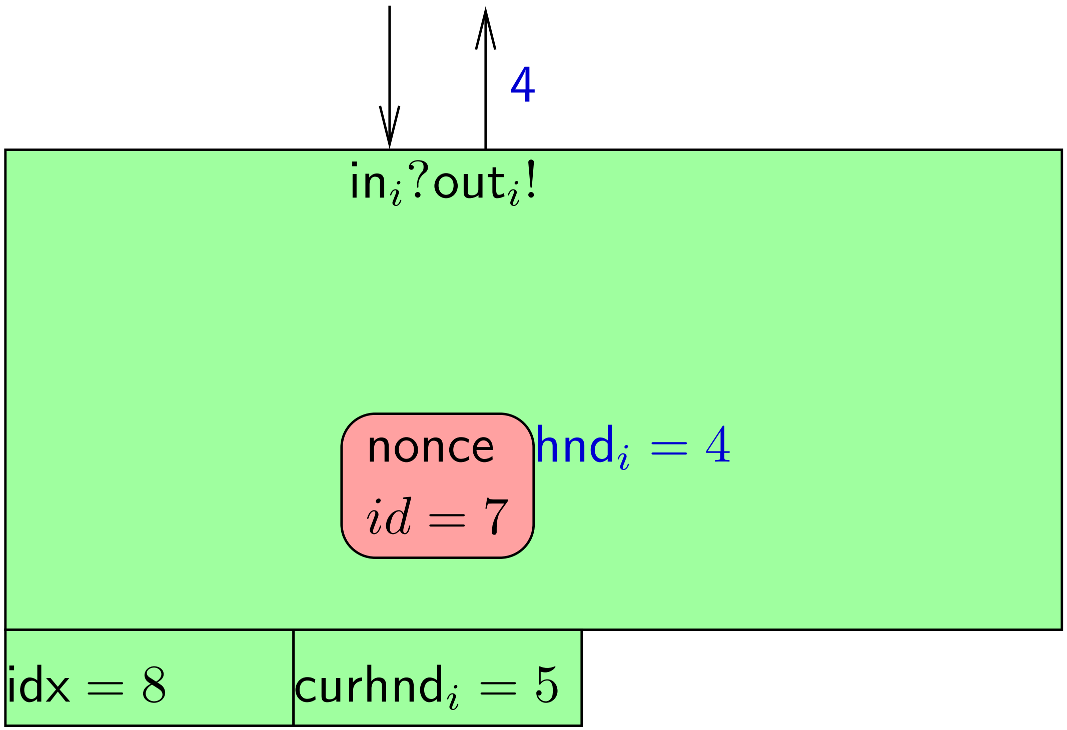
Operations: example 1



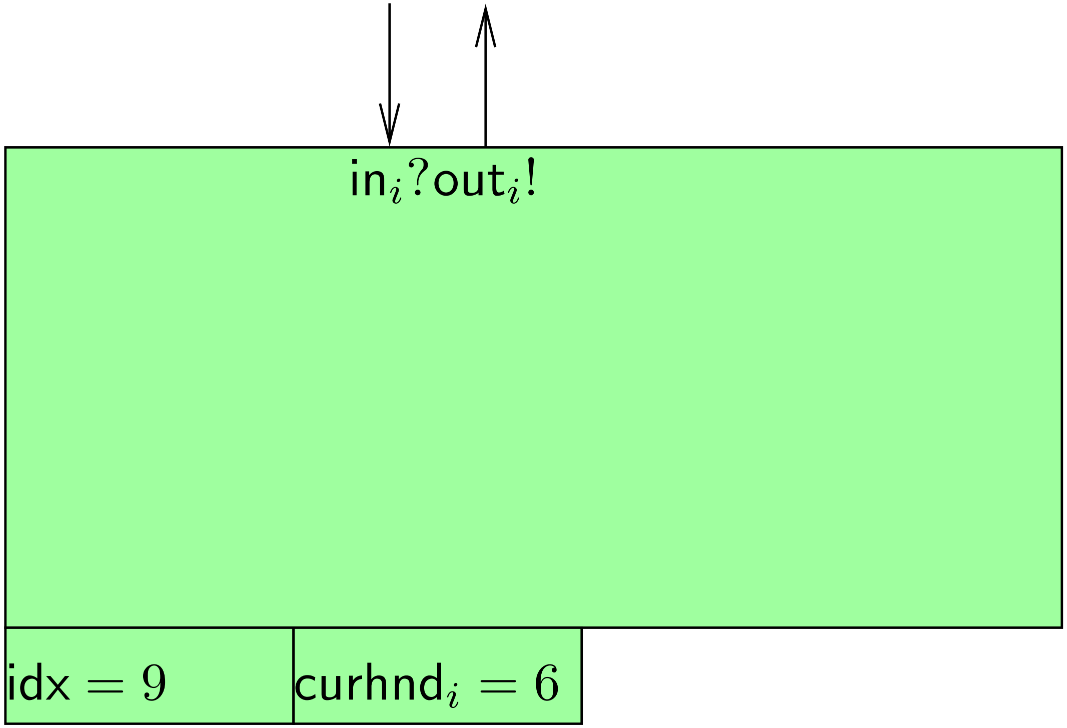
Operations: example 1



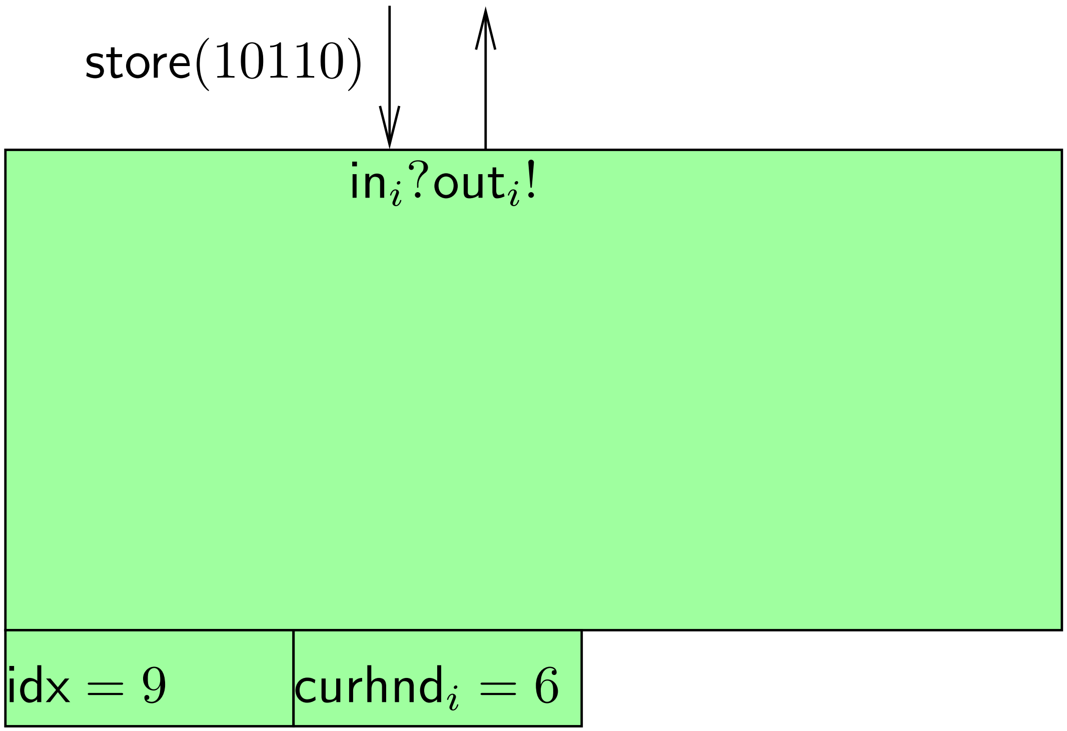
Operations: example 1



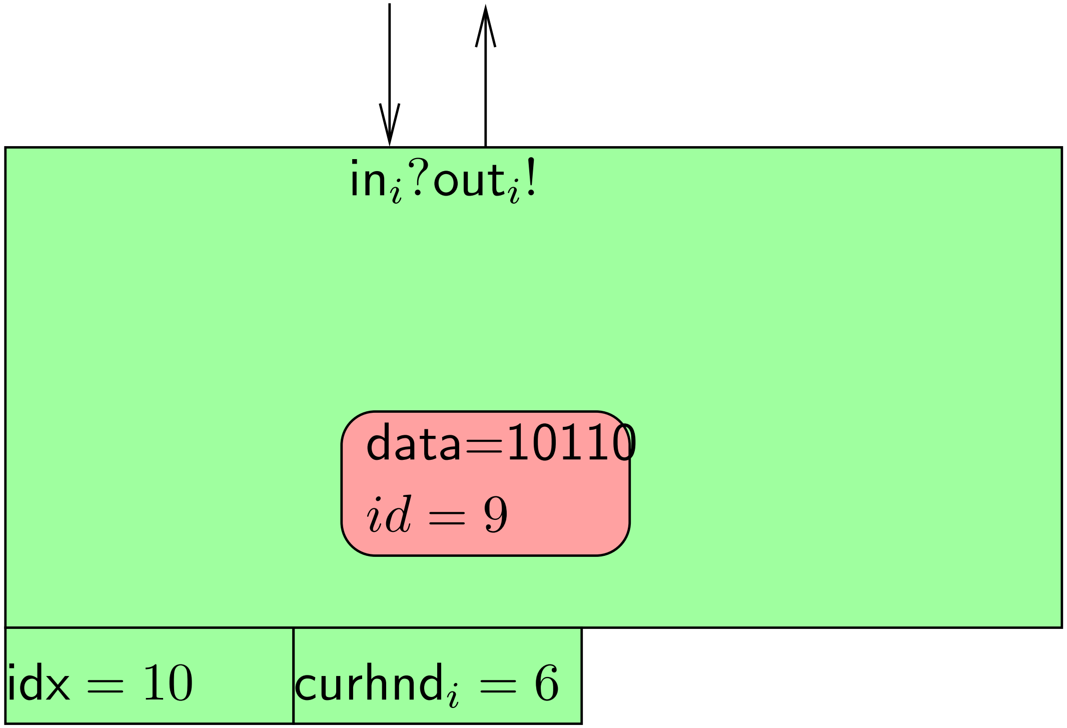
Operations: example 2



Operations: example 2



Operations: example 2



in$_i$?out$_i$!

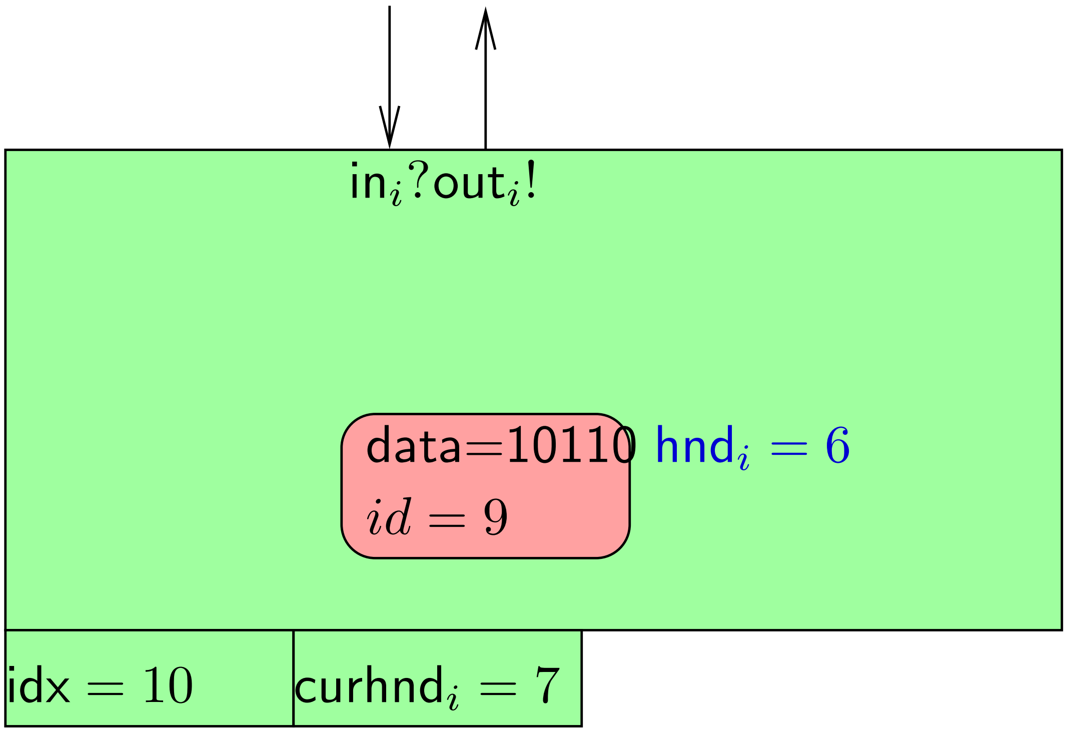
data=10110

$id = 9$

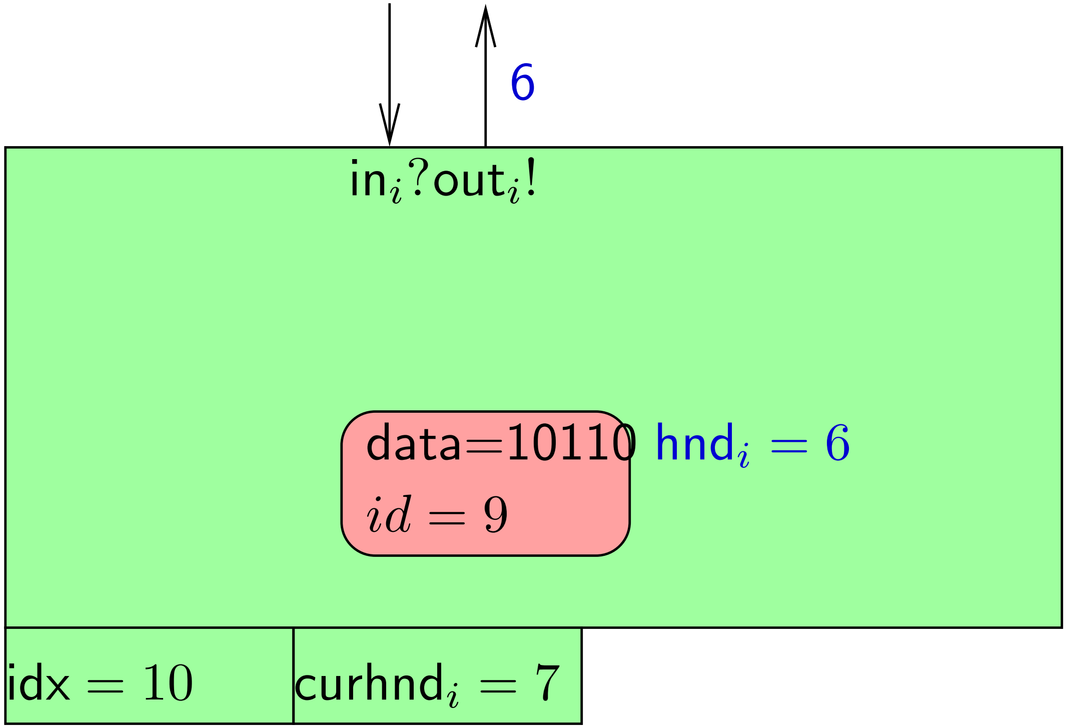
idx $= 10$

curhnd$_i$ $= 6$

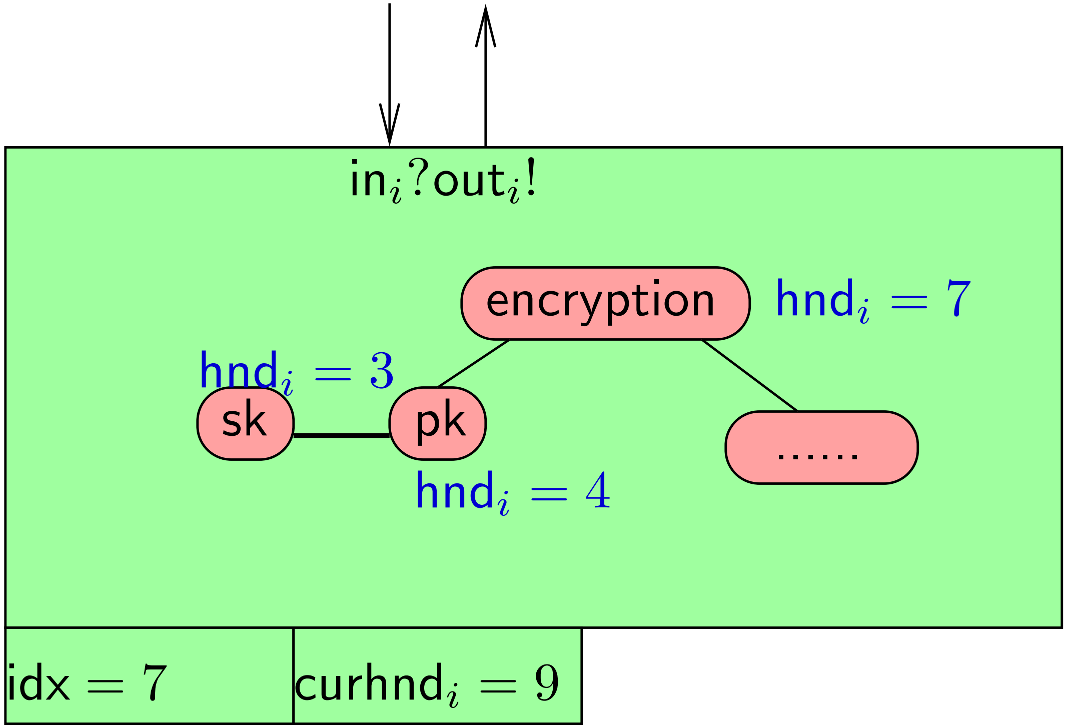
Operations: example 2



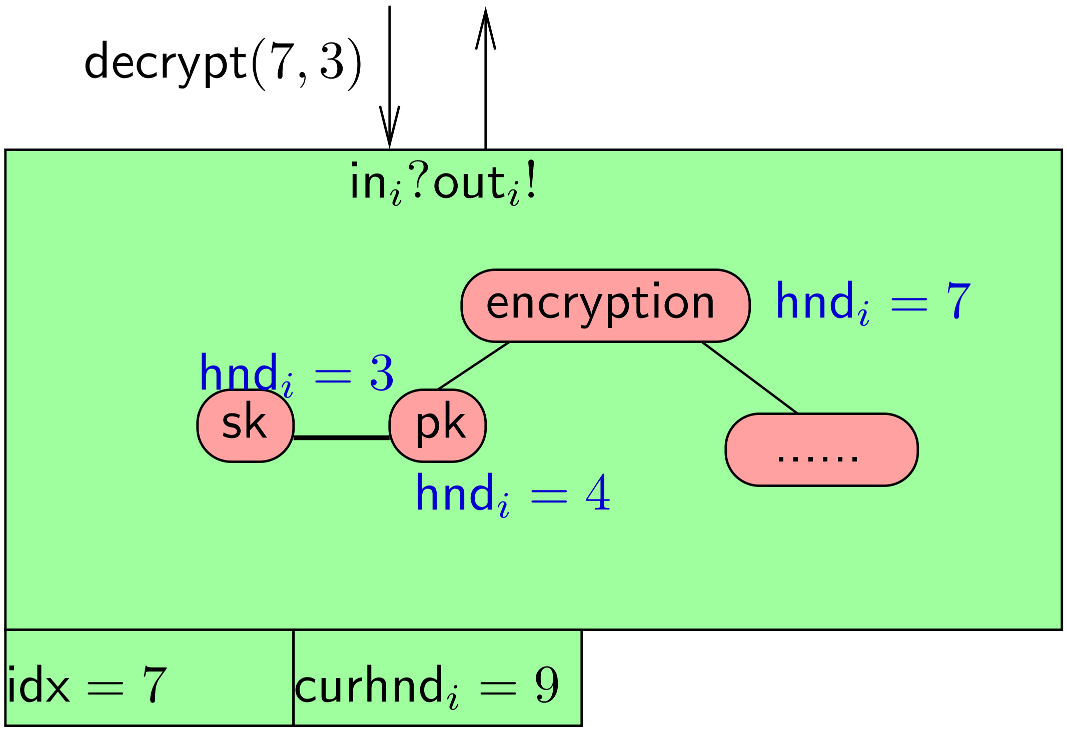
Operations: example 2


6
$\mathsf{in}_i?\mathsf{out}_i!$
data=10110 $\mathsf{hnd}_i = 6$
$id = 9$
$\mathsf{idx} = 10$
$\mathsf{curhnd}_i = 7$

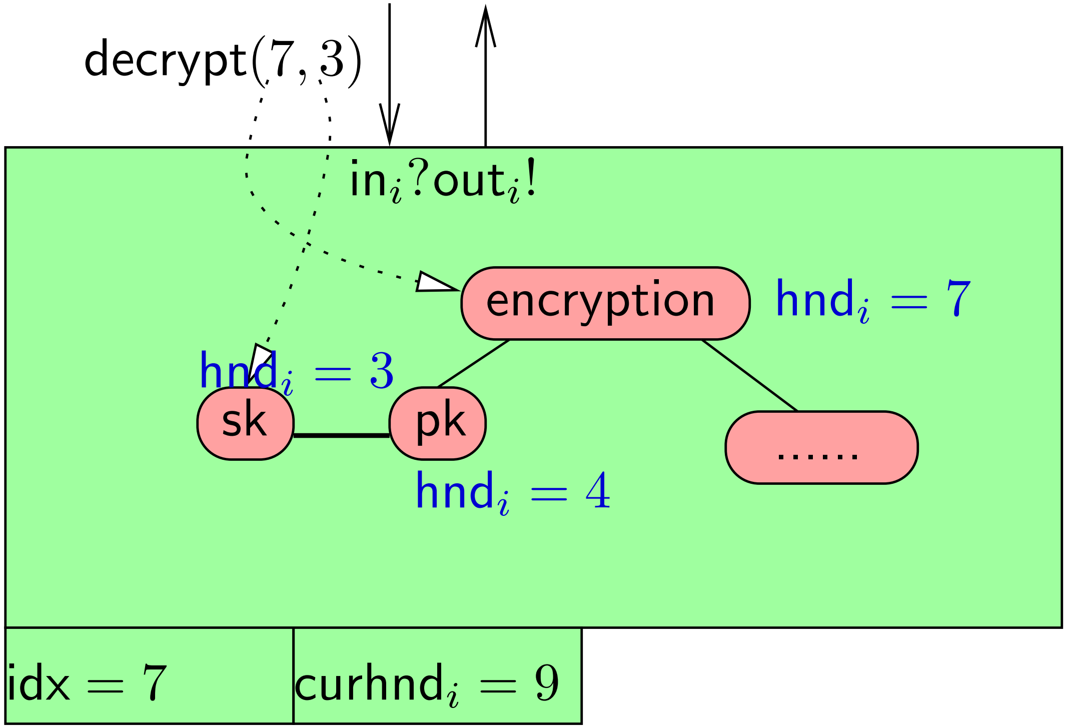
Operations: example 3



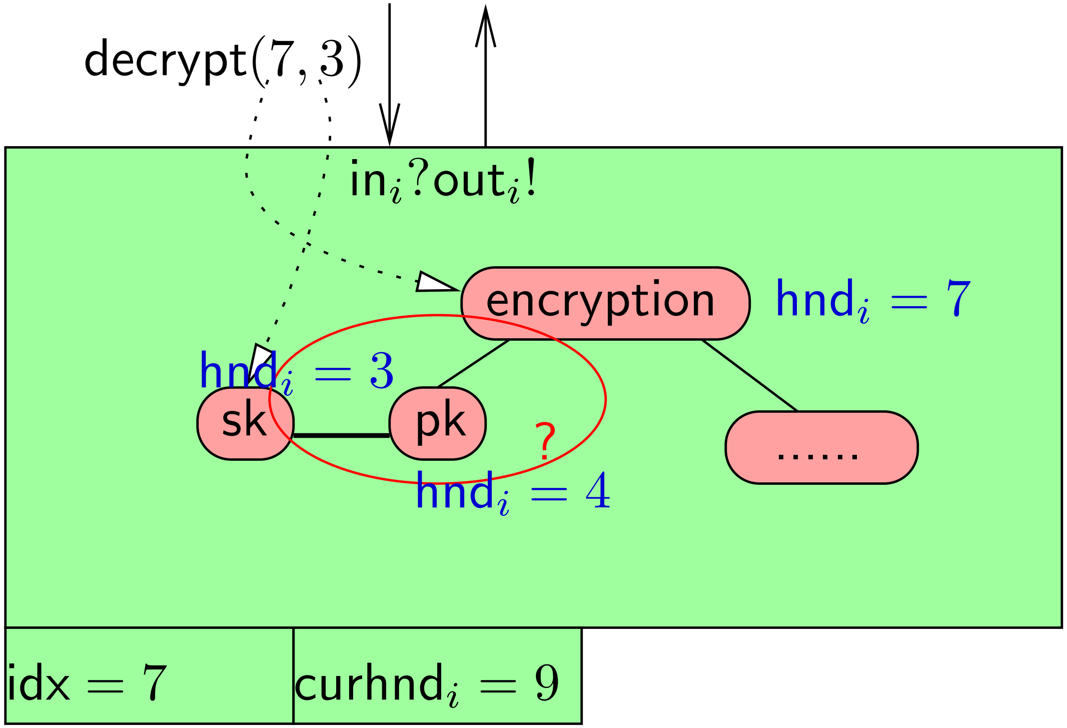
Operations: example 3



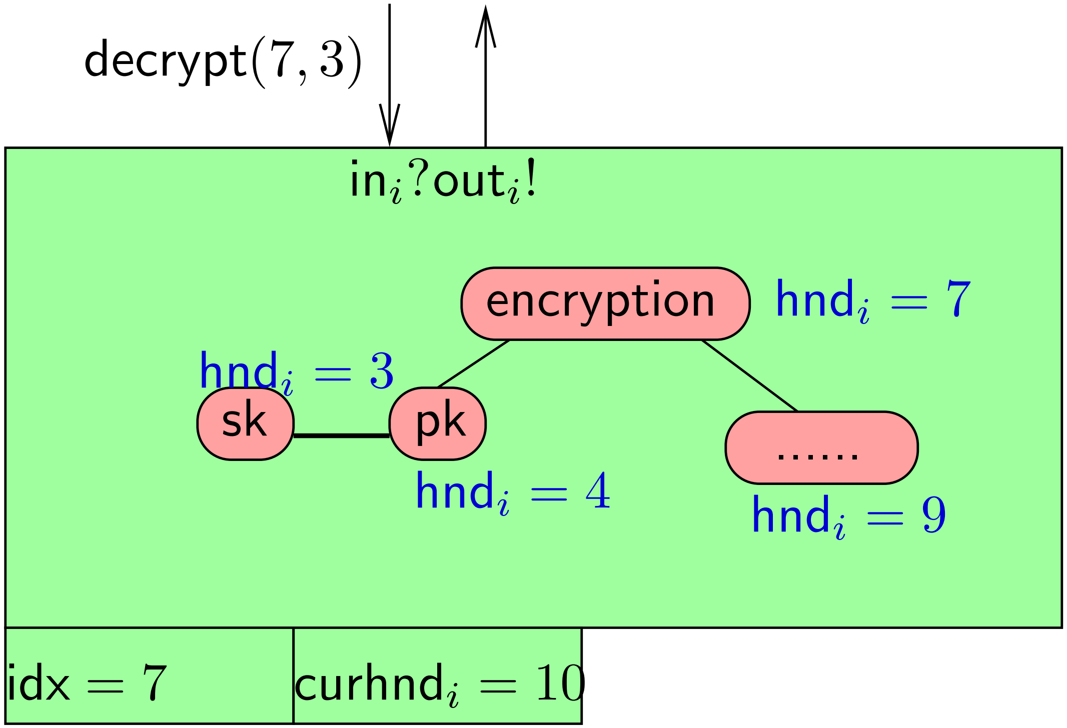
Operations: example 3



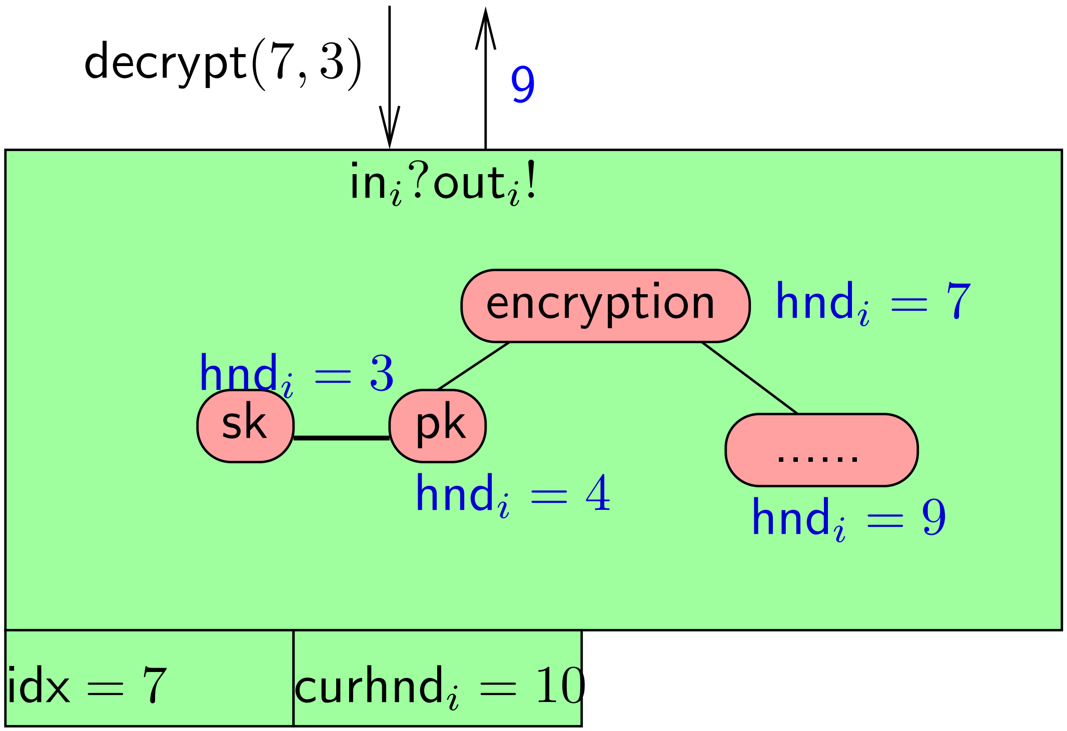
Operations: example 3



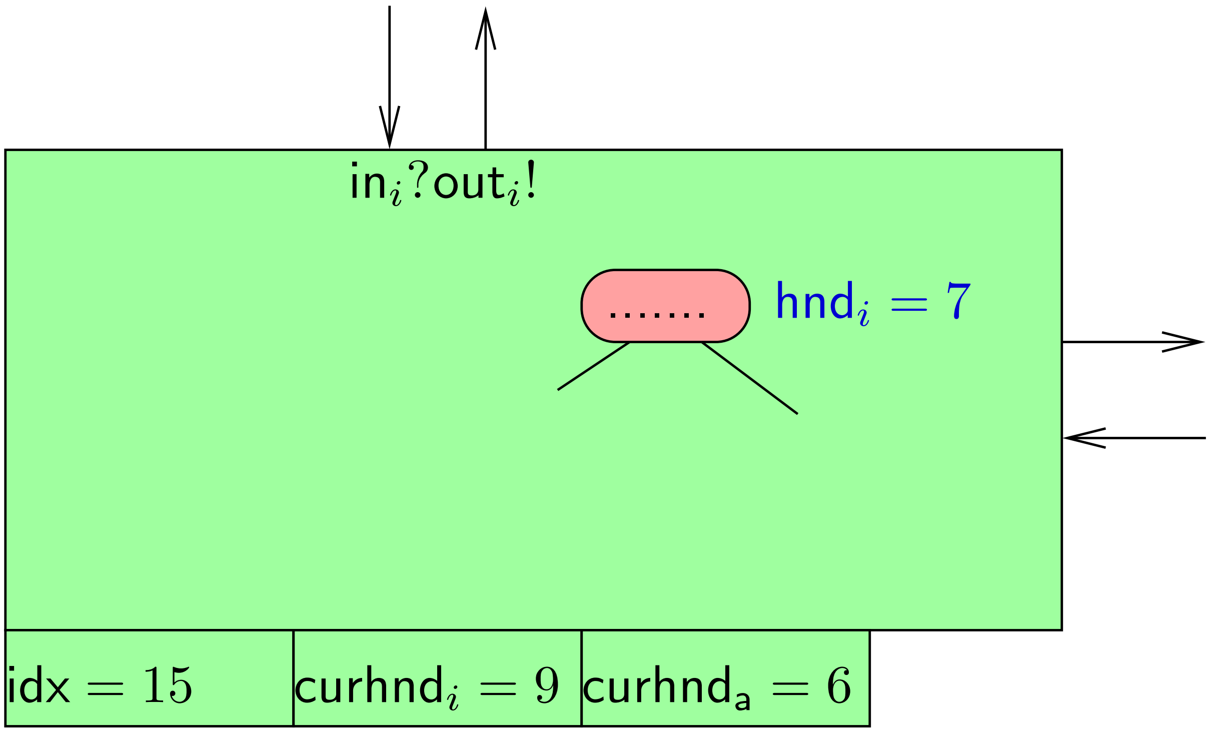
Operations: example 3



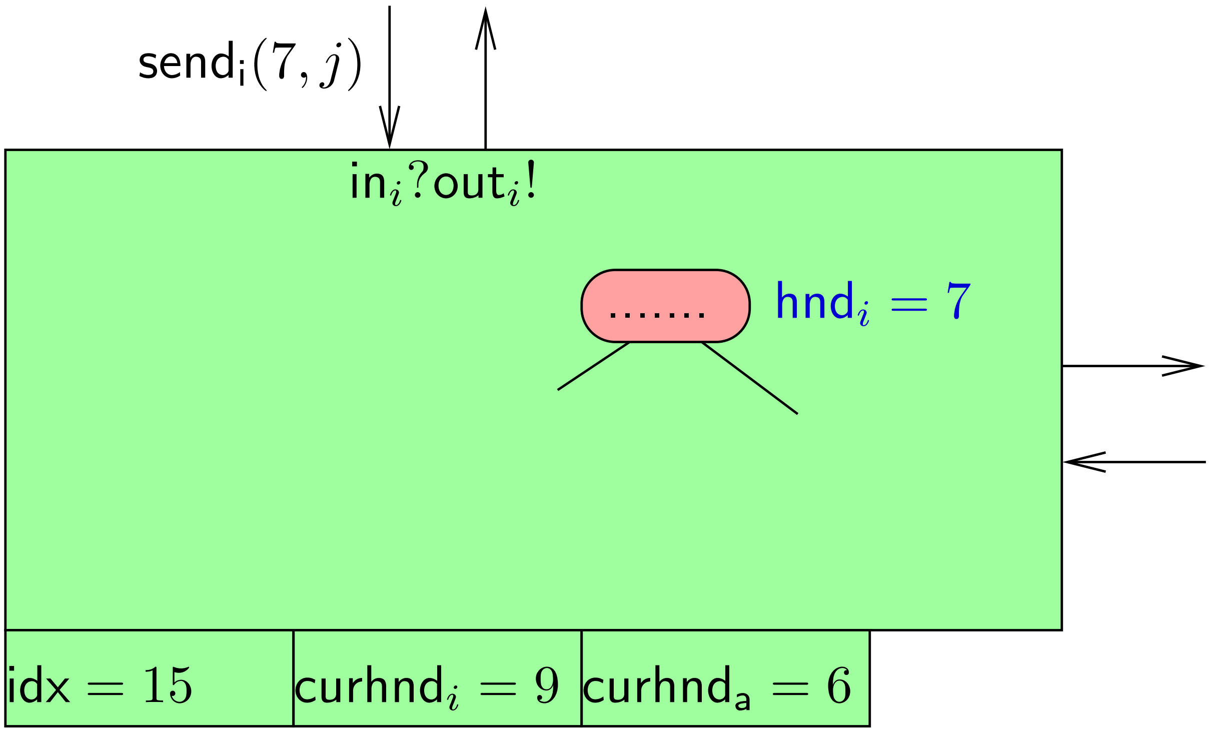
Operations: example 3



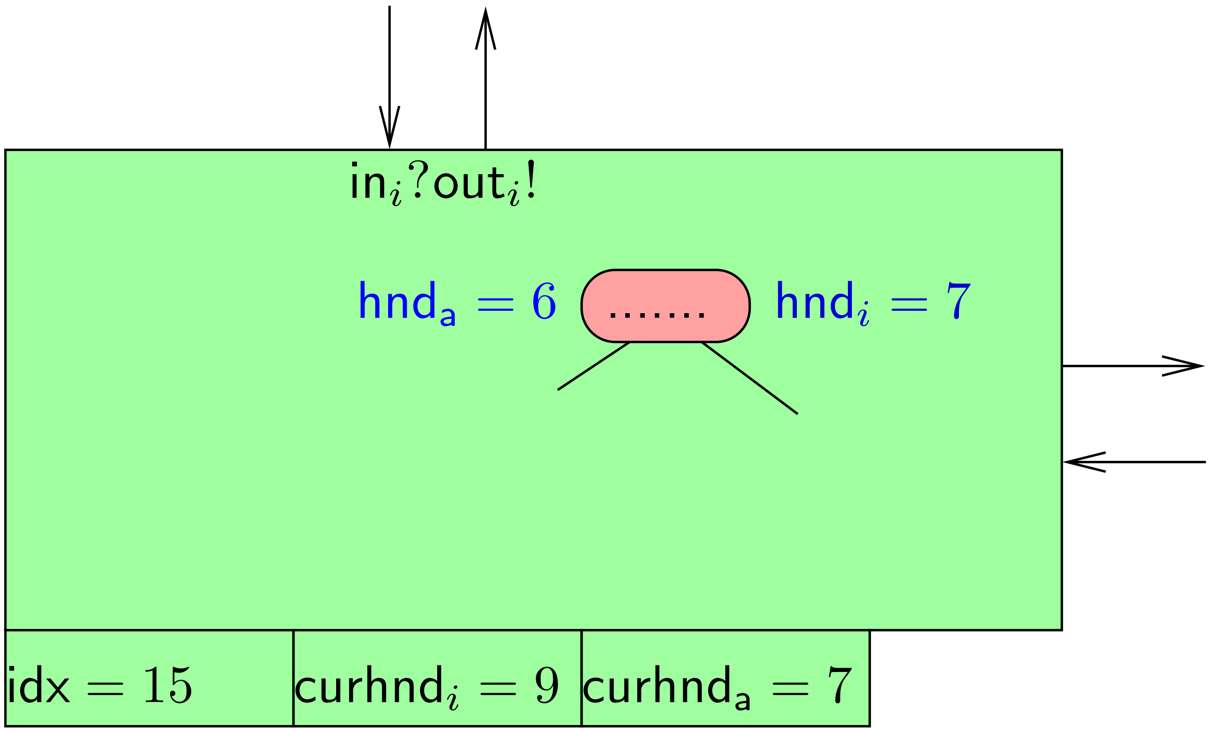
Operations: example 4



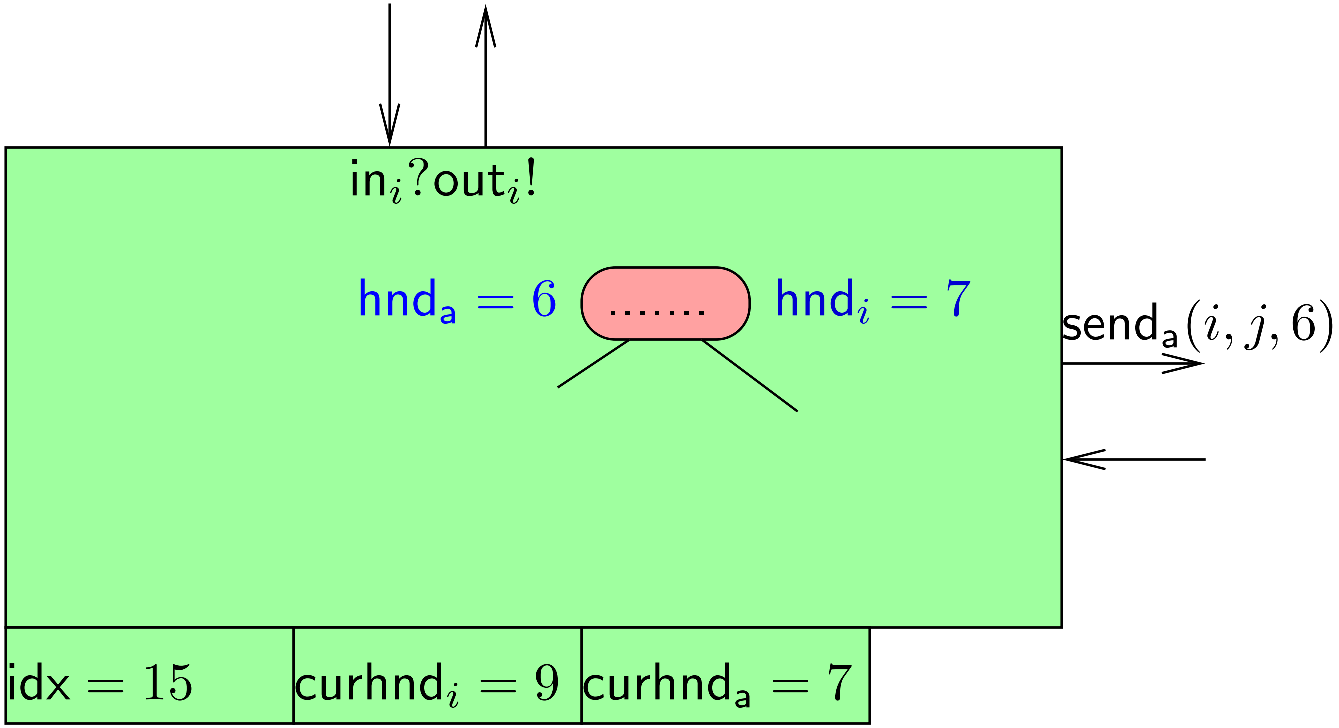
Operations: example 4



Operations: example 4

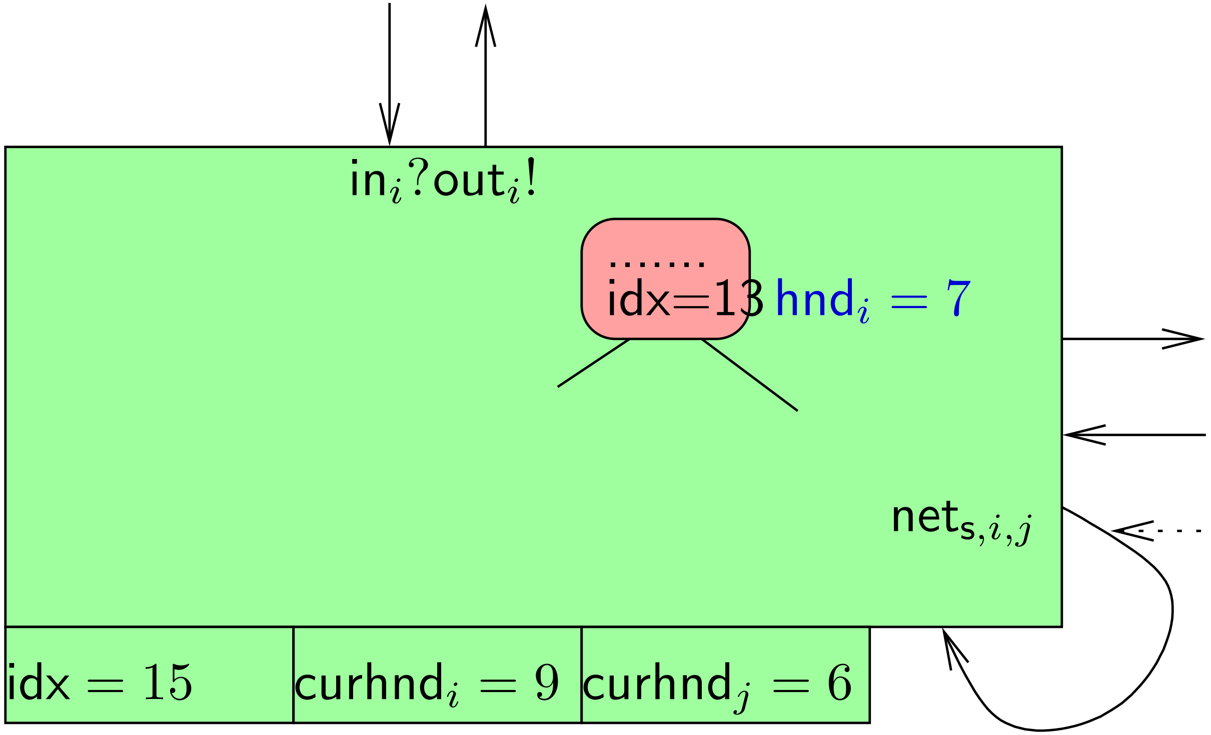

$\mathsf{in}_i?\mathsf{out}_i!$
$\mathsf{hnd}_\mathsf{a} = 6$
....
$\mathsf{hnd}_i = 7$
$\mathsf{idx} = 15$
$\mathsf{curhnd}_i = 9$
$\mathsf{curhnd}_\mathsf{a} = 7$

Operations: example 4

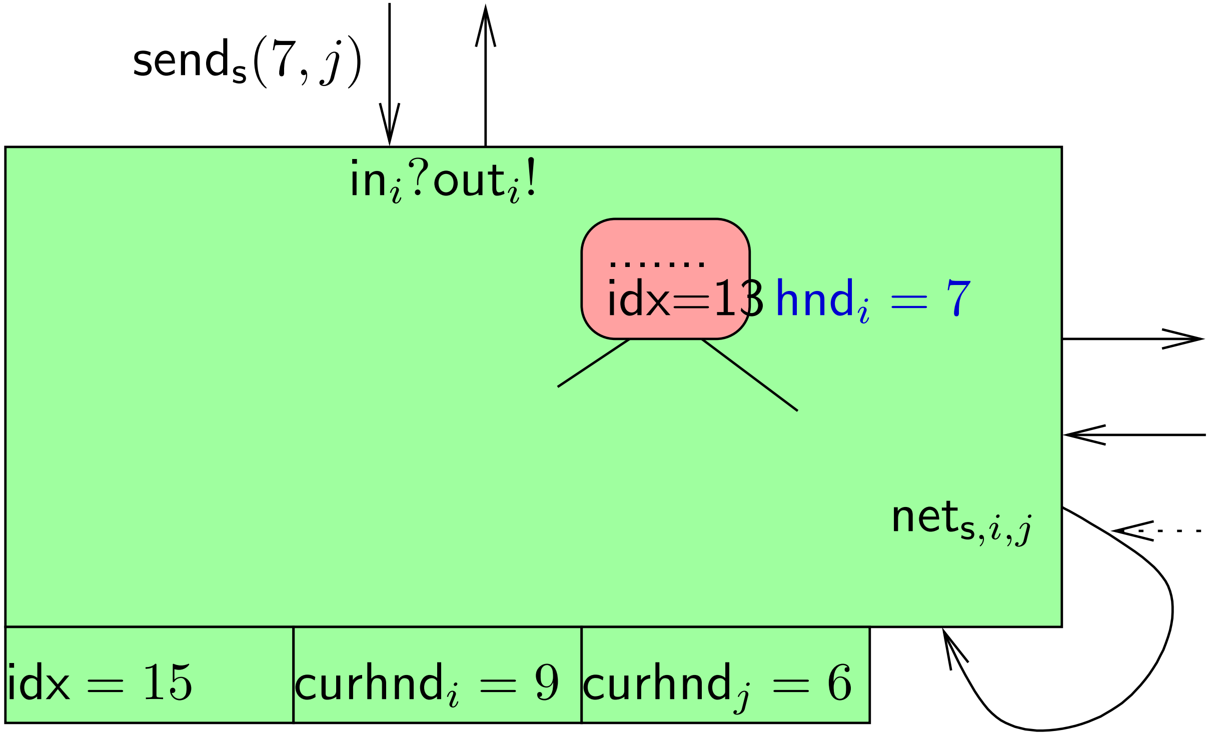


$\mathsf{in}_i?\mathsf{out}_i!$
$\mathsf{hnd}_\mathsf{a} = 6$
....
$\mathsf{hnd}_i = 7$
$\mathsf{send}_\mathsf{a}(i, j, 6)$
$\mathsf{idx} = 15$
$\mathsf{curhnd}_i = 9$
$\mathsf{curhnd}_\mathsf{a} = 7$

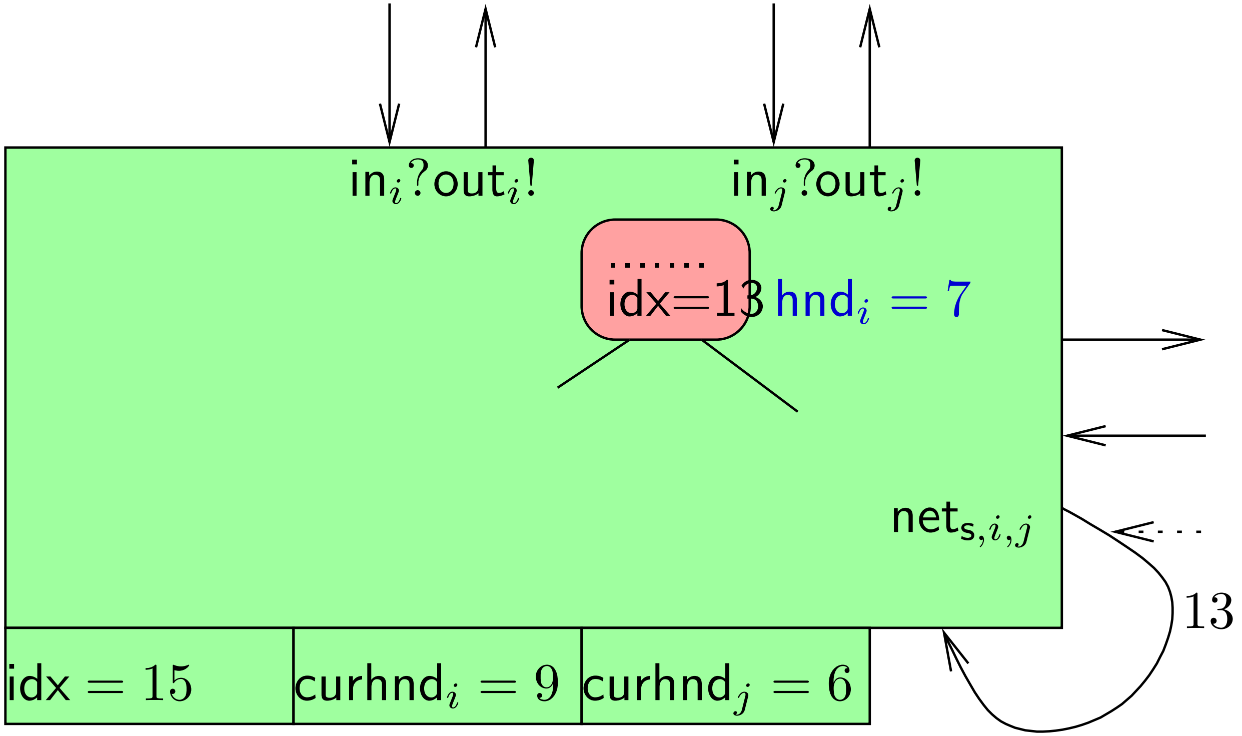
Operations: example 5



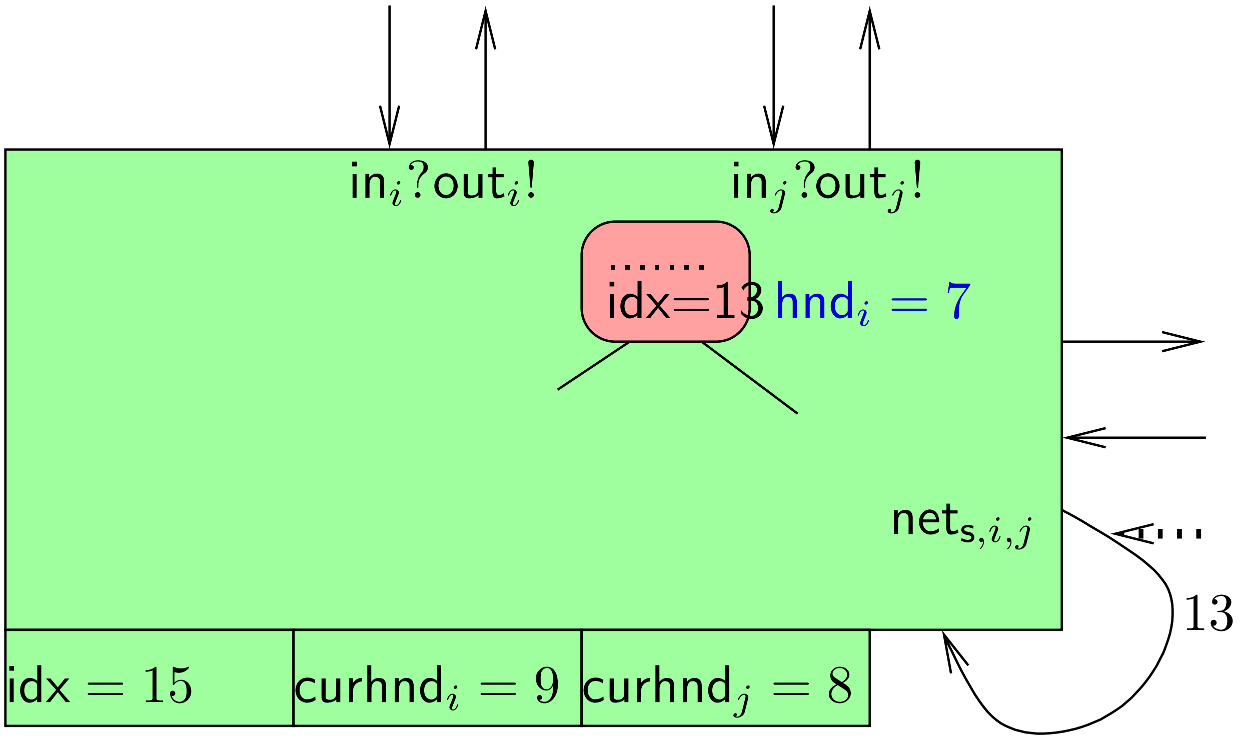
Operations: example 5



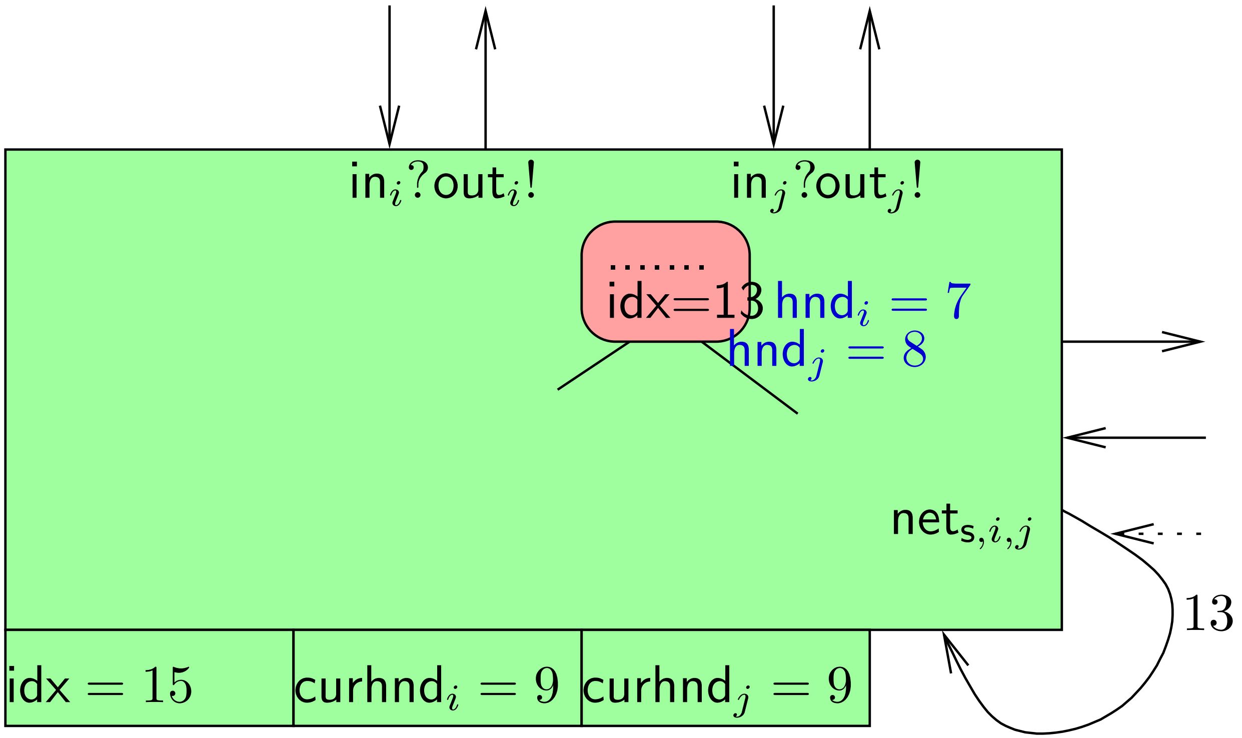
Operations: example 5



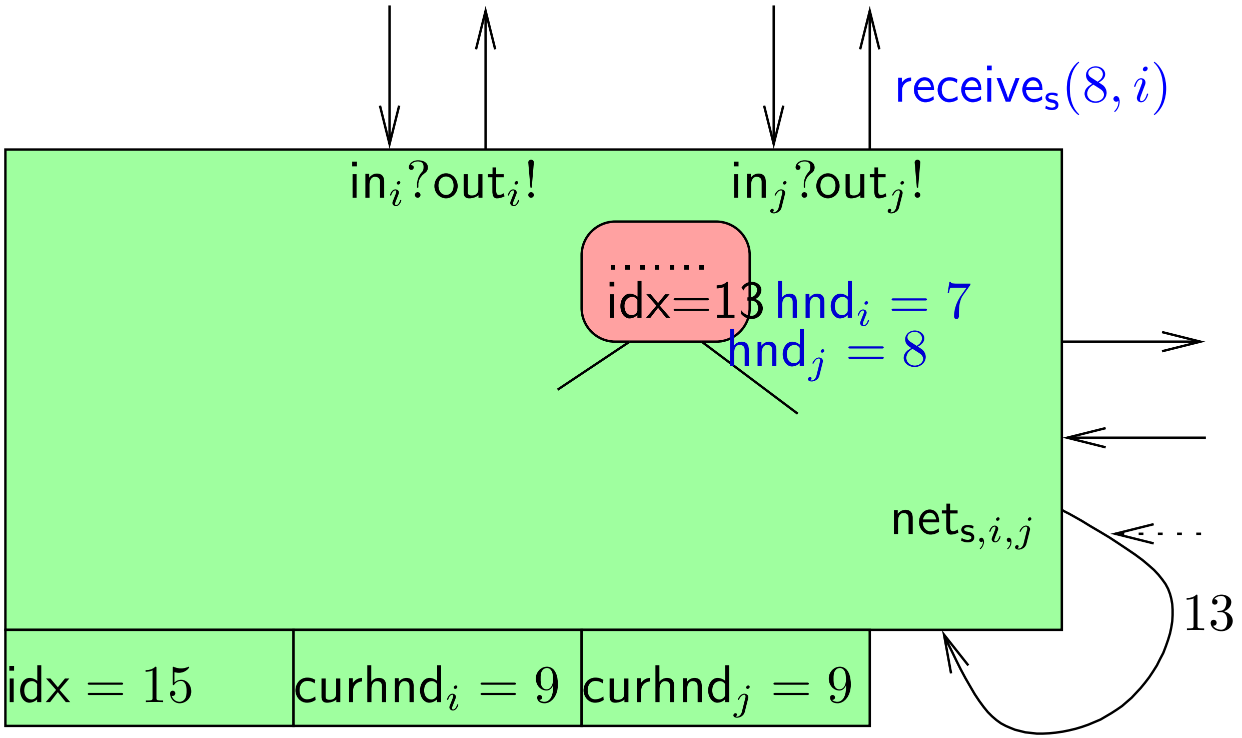
Operations: example 5



Operations: example 5

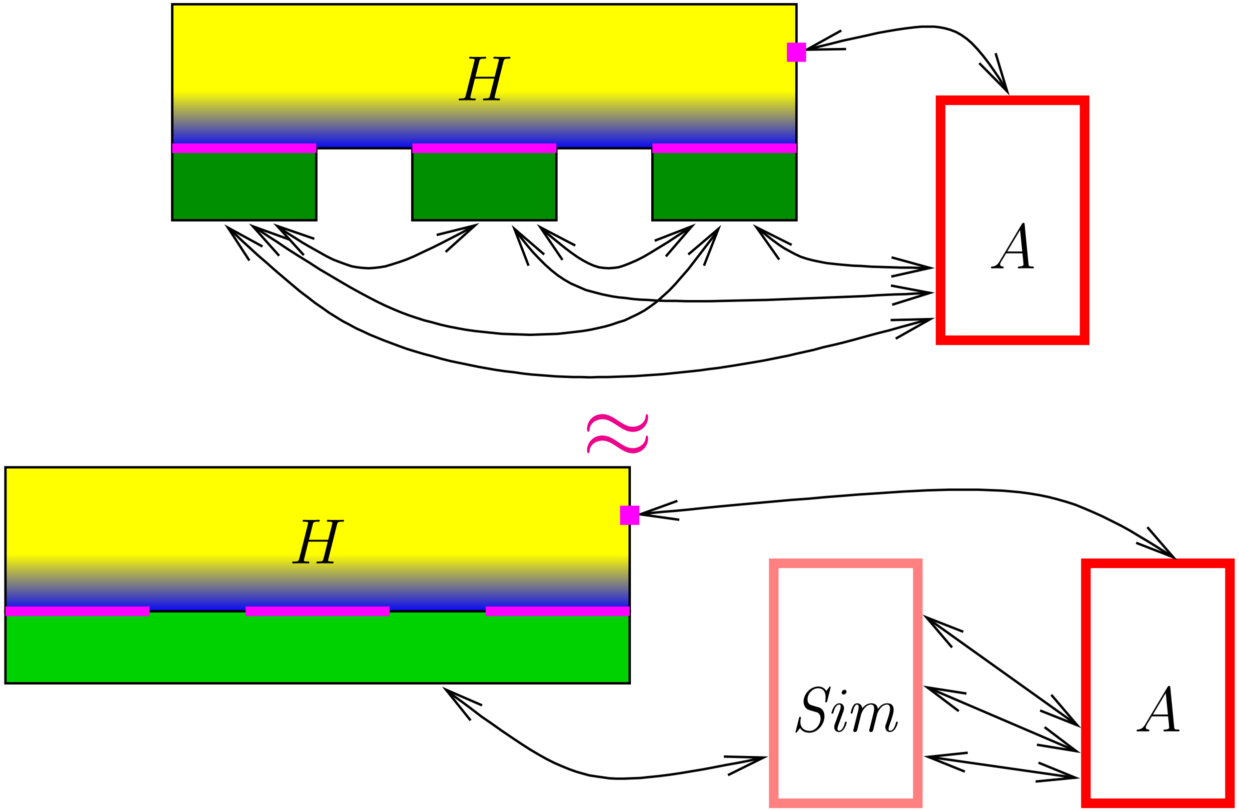


Operations: example 5



Simulatability

$\exists\, Sim$, such that for all A and almost all H:



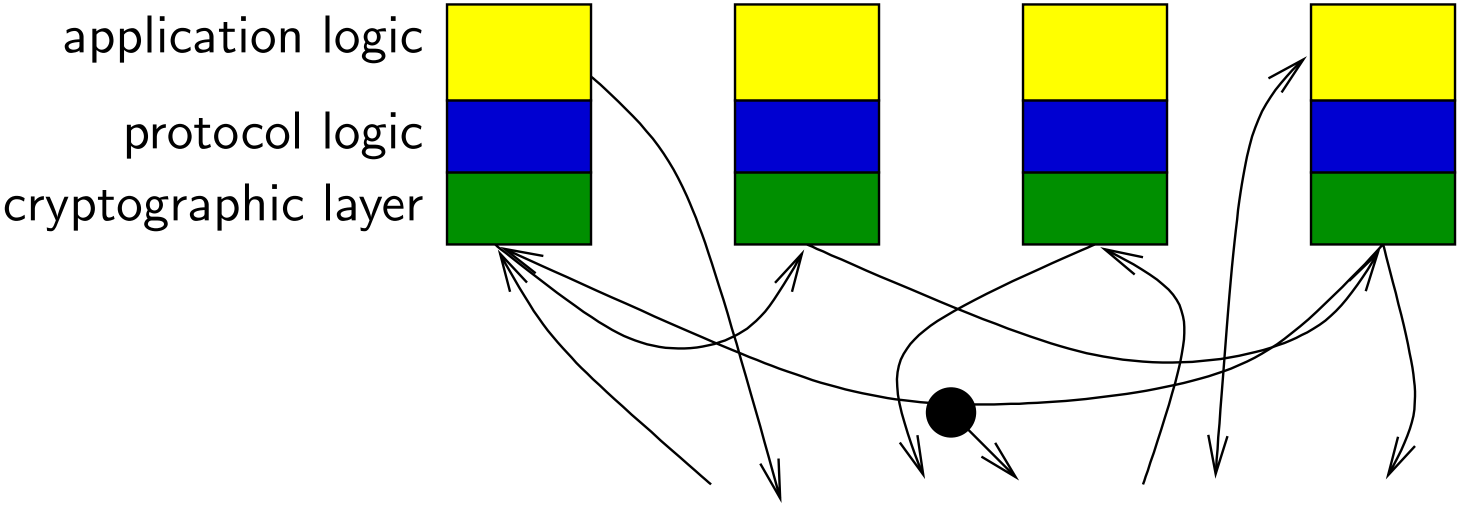
- The views of the user H must be indistinguishable.
- Conditions on H nontrivial, but not too restrictive.

Simulatability means. . .

- We say that the real library is at least as secure as the ideal library.
- Meaning of the definition: anything that may happen to the user of the concrete library may also happen to the user of the abstract library.
  - this “anything” includes all bad things.
- Vice versa: if nothing bad can happen to the user of the abstract library then nothing bad can happen to the user of the concrete library.

In our case. . .

instead of analysing

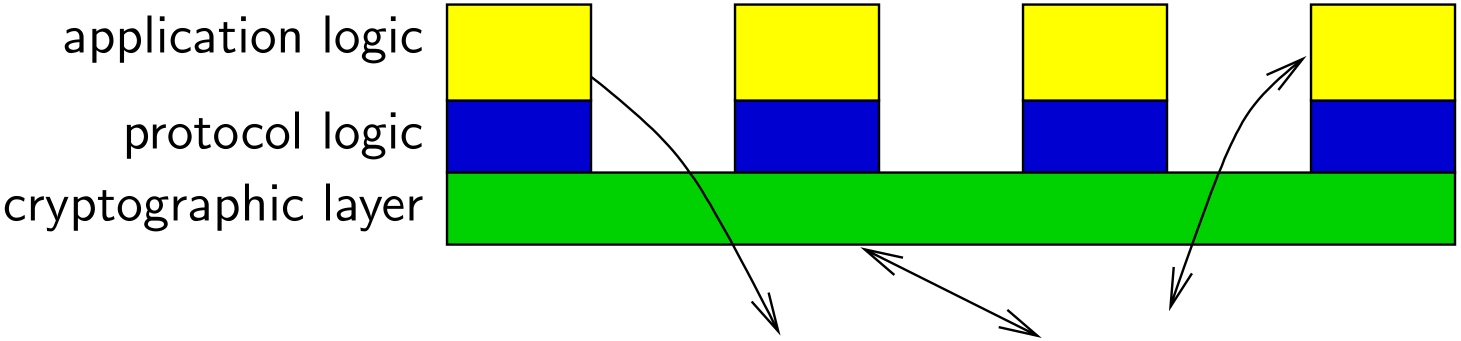


application logic

protocol logic

cryptographic layer

we may analyse



application logic

protocol logic

cryptographic layer

and this is most likely easier.

Offered primitives

- The library currently offers
  - symmetric encryption;
  - asymmetric encryption;
  - signatures;
  - message authentication codes;
  - (in random oracle model: hash functions).
- There are other primitives that are used in many interesting protocols

Offered primitives

- The library currently offers
  - symmetric encryption;
  - asymmetric encryption;
  - signatures;
  - message authentication codes;
  - (in random oracle model: hash functions).
- There are other primitives that are used in many interesting protocols
- For example, homomorphic encryption

Homomorphic encryption

- Asymmetric encryption, given by algorithms $\mathcal{K}$, $\mathcal{E}$, $\mathcal{D}$.
- Security — IND-CPA (as usual)

IND-CPA security

- Consider the following game (against an adversary):
  - Generate a public key pk.
    - The secret key is unnecessary in this game
  - Give pk to the adversary.
  - The adversary submits two plaintexts m_0, m_1 of equal length.
  - Generate random bit b, give $\mathcal{E}(pk, m_b)$ to the adversary.
  - The adversary comes up with a guess b^* for b.
- Encryption scheme is IND-CPA-secure, if no efficient adversary can guess b with probability significantly larger than $1/2$.

Homomorphic encryption

- Asymmetric encryption, given by algorithms $\mathcal{K}$, $\mathcal{E}$, $\mathcal{D}$.
- Security — IND-CPA (as usual)
- Set of possible plaintexts must be Abelian group.
- For any keypair (pk, sk) and plaintexts x, x', the following must with overwhelming probability:

$$\mathcal{D}(sk, \mathcal{E}(pk, x) \odot \mathcal{E}(pk, x')) = x + x'$$

for some operation $\odot$ on ciphertexts.

Homomorphic encryption

- Asymmetric encryption, given by algorithms $\mathcal{K}$, $\mathcal{E}$, $\mathcal{D}$.
- Security — IND-CPA (as usual)
- Set of possible plaintexts must be Abelian group.
- For any keypair (pk, sk) and plaintexts x, x', the following must with overwhelming probability:

$$\mathcal{D}(sk, \mathcal{E}(pk, x) \odot \mathcal{E}(pk, x')) = x + x'$$

  for some operation $\odot$ on ciphertexts.
- Useful in auctions, e-voting, data mining, etc.

t-out-of-n threshold encryption

- Algorithms:
  - Key generation $\mathcal{K}$ returns pk, $sk_1, \ldots, sk_n$, $vk_1, \ldots, vk_n$.
  - Encryption $\mathcal{E}$ works as usual.
  - Decryption $\mathcal{D}(sk_i, c)$ returns the plaintext share ds_i and its correctness proof dp_i.
  - Share verification $\mathcal{V}(vk_i, c, ds_i, dp_i)$ allows to verify the correctness of decryption.
  - Share combination $\mathcal{C}(ds_{i_1}, \ldots, ds_{i_t})$ combines the shares into the plaintext.
- Allows the distribution of authorities.

Putting it together

Threshold homomorphic encryption!

- Security: IND-CPA even after the adversary has learned up to $t-1$ secret key shares.
  - There must exist a simulation algorithm $\mathcal{S}$, such that $\mathcal{S}(m, c, ds_{i_1}, \ldots, ds_{i_u})$, where $u \leq t-1$, returns $ds_1, \ldots, ds_n$, such that
    - any t of them combine to m;
    - the returned ds_j is indistinguishable from the real share to someone who knows $sk_{i_1}, \ldots, sk_{i_u}$.

Non-interactive zero-knowledge proofs

- Let $\mathcal{L}$ be a language in NP.
  - Let R bet its witness relation.
    - $x \; R \; w$ is decidable in polynomial time.
    - $x \in \mathcal{L}$ iff $\exists w : x \; R \; w$ and $|w|$ is polynomial in $|x|$.
- A NIZK proof system for R is a pair of algorithms:
  - $\overline{\mathcal{P}}(x, w)$ returns the proof of knowledge π of w;
  - $\overline{\mathcal{V}}(x, \pi)$ verifies the given proof of knowledge wrt. x.
- Security properties:
  - π does not leak anything about w;
  - an accepted π can only be constructed with the knowledge of w.

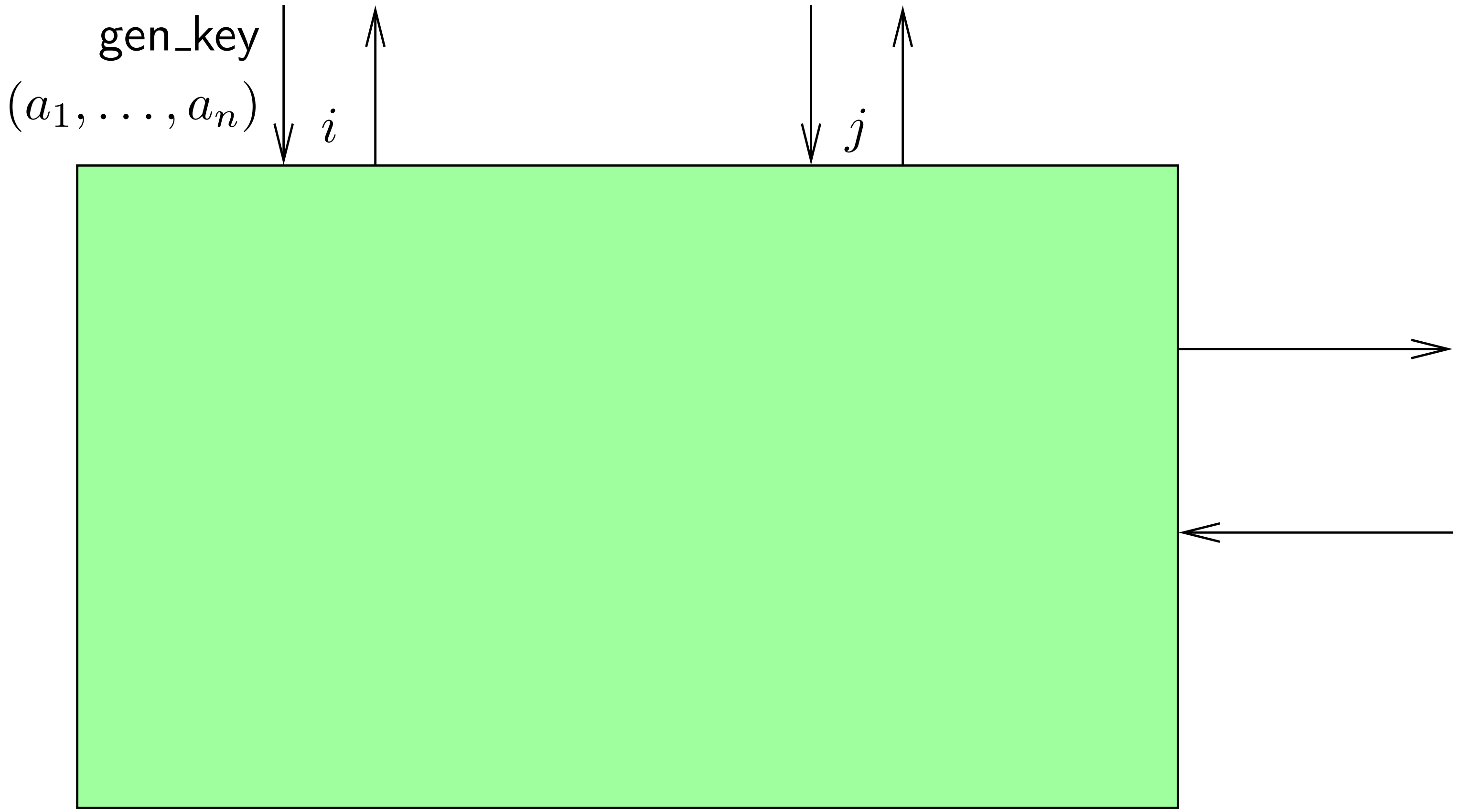
Non-interactive zero-knowledge proofs

- Let $\mathcal{L}$ be a language in NP.
  - Let R bet its witness relation.
    - $x\ R\ w$ is decidable in polynomial time.
    - $x \in \mathcal{L}$ iff $\exists w : x\ R\ w$ and $|w|$ is polynomial in $|x|$.
- A NIZK proof system for R is a pair of algorithms:
  - $\overline{\mathcal{P}}(x, w)$ returns the proof of knowledge π of w;
  - $\overline{\mathcal{V}}(x, \pi)$ verifies the given proof of knowledge wrt. x.
- Security properties:
  - π does not leak anything about w;
  - an accepted π can only be constructed with the knowledge of w.
- Example: showing that the plaintext corresponding to the ciphertext c satisfies some property.

T.H.E. in the abstract library

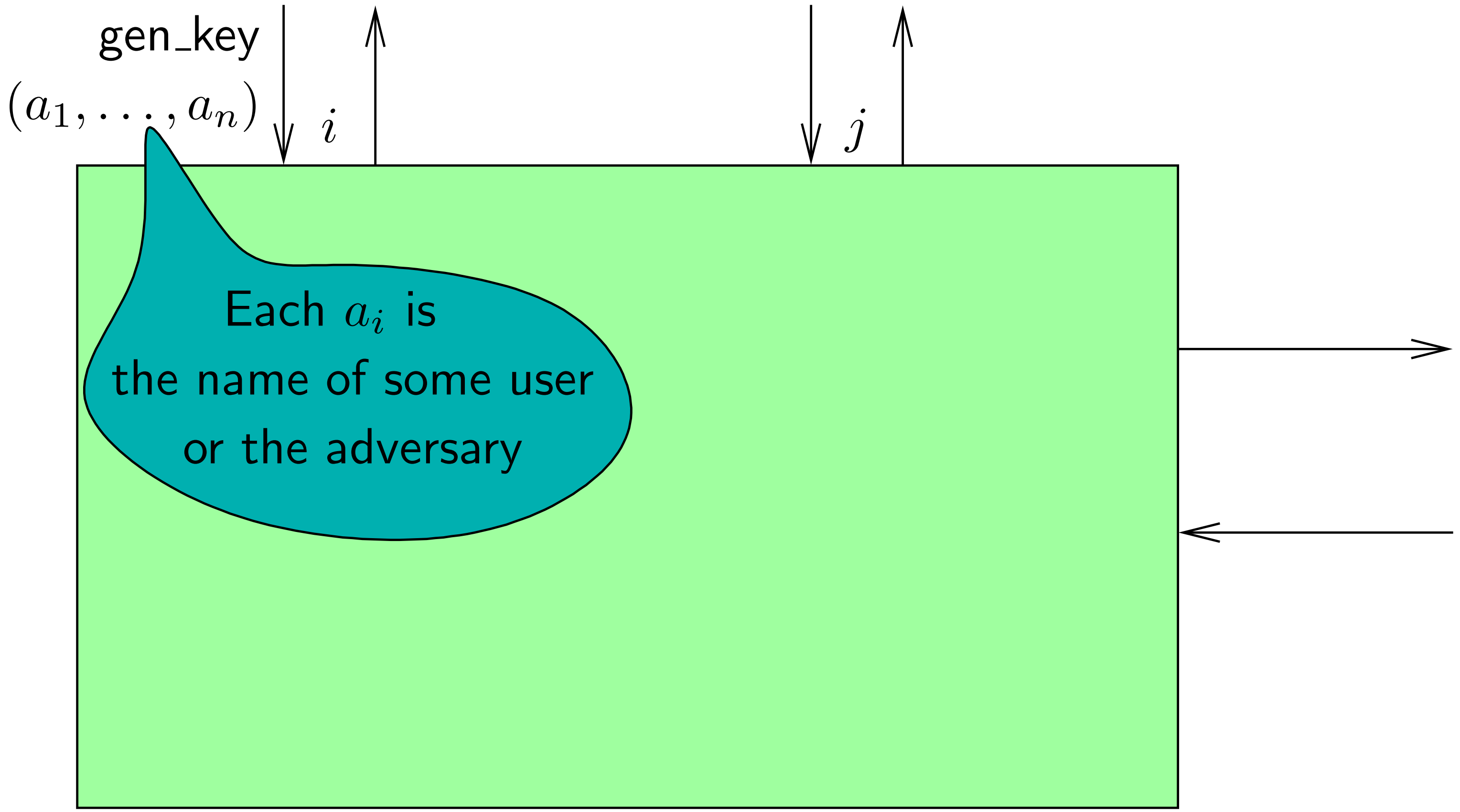
Abstract Library: key generation

- (Start to) generate a new set of keys
  - Specify the recipient of each secret key share



Abstract Library: key generation

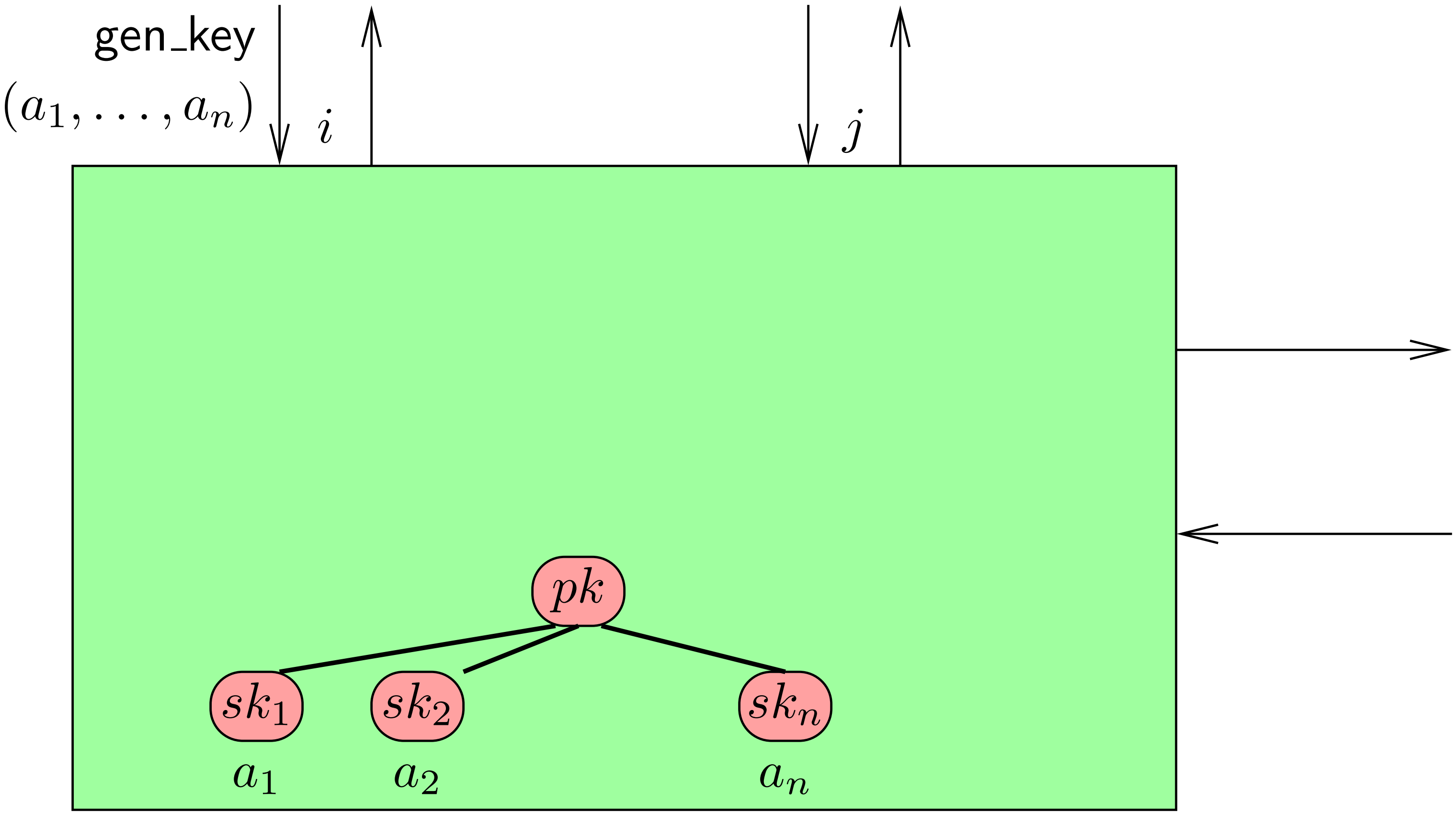
- (Start to) generate a new set of keys
  - Specify the recipient of each secret key share



Each a_i is the name of some user or the adversary

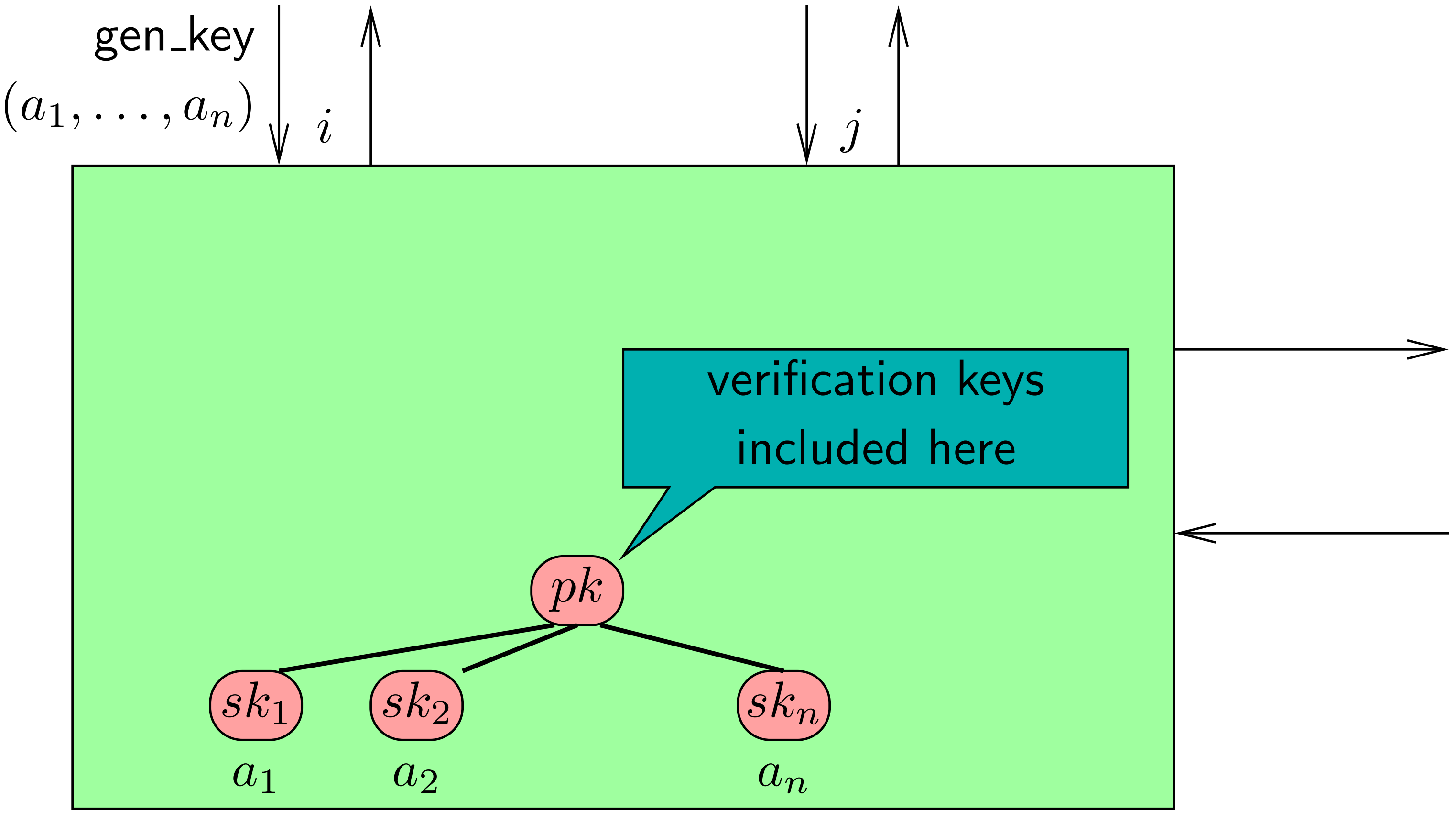
Abstract Library: key generation

- (Start to) generate a new set of keys
  - Specify the recipient of each secret key share



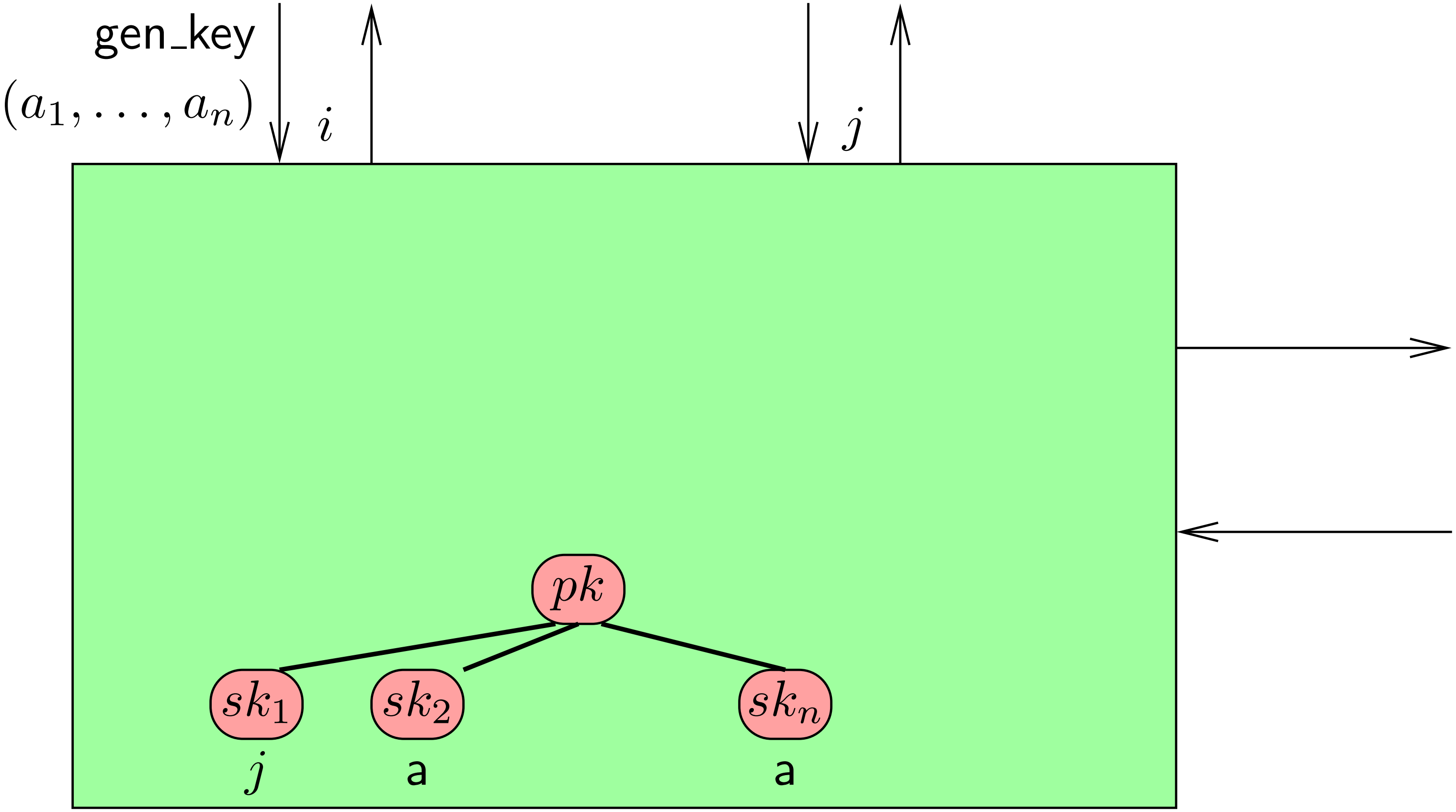
Abstract Library: key generation

- (Start to) generate a new set of keys
  - Specify the recipient of each secret key share



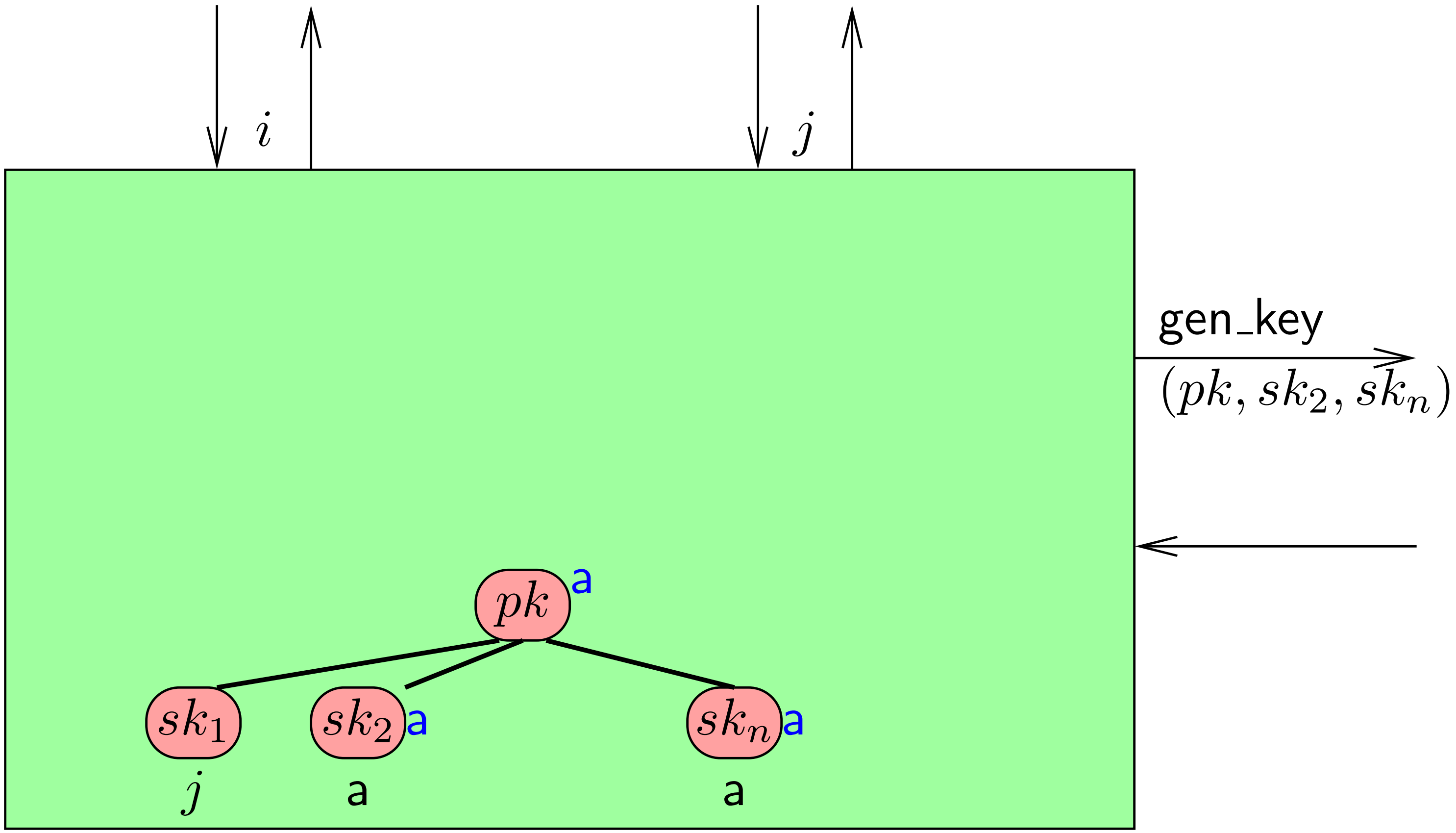
Abstract Library: key generation

- (Start to) generate a new set of keys
  - Specify the recipient of each secret key share



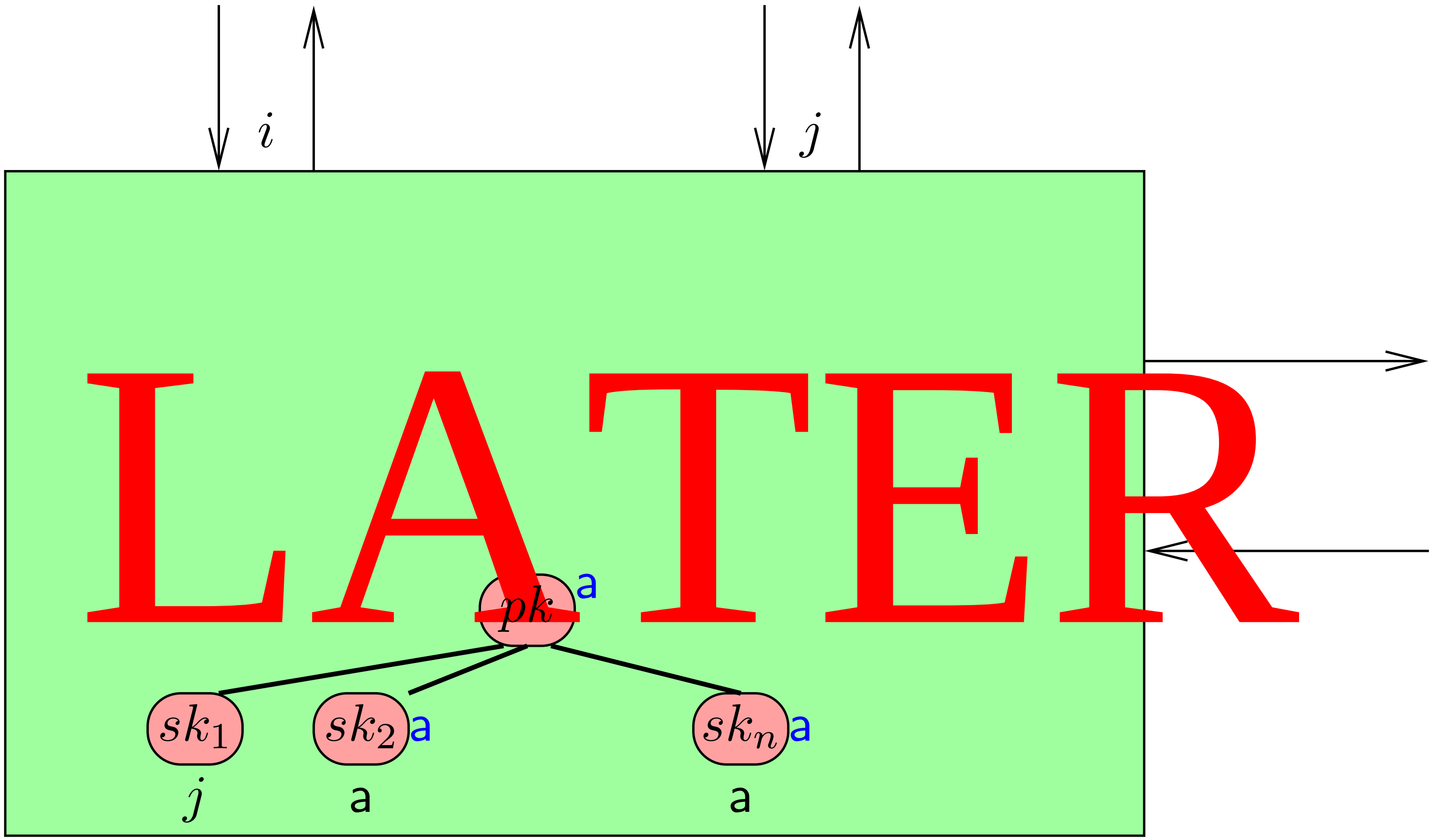
Abstract Library: key generation

- (Start to) generate a new set of keys
  - Specify the recipient of each secret key share



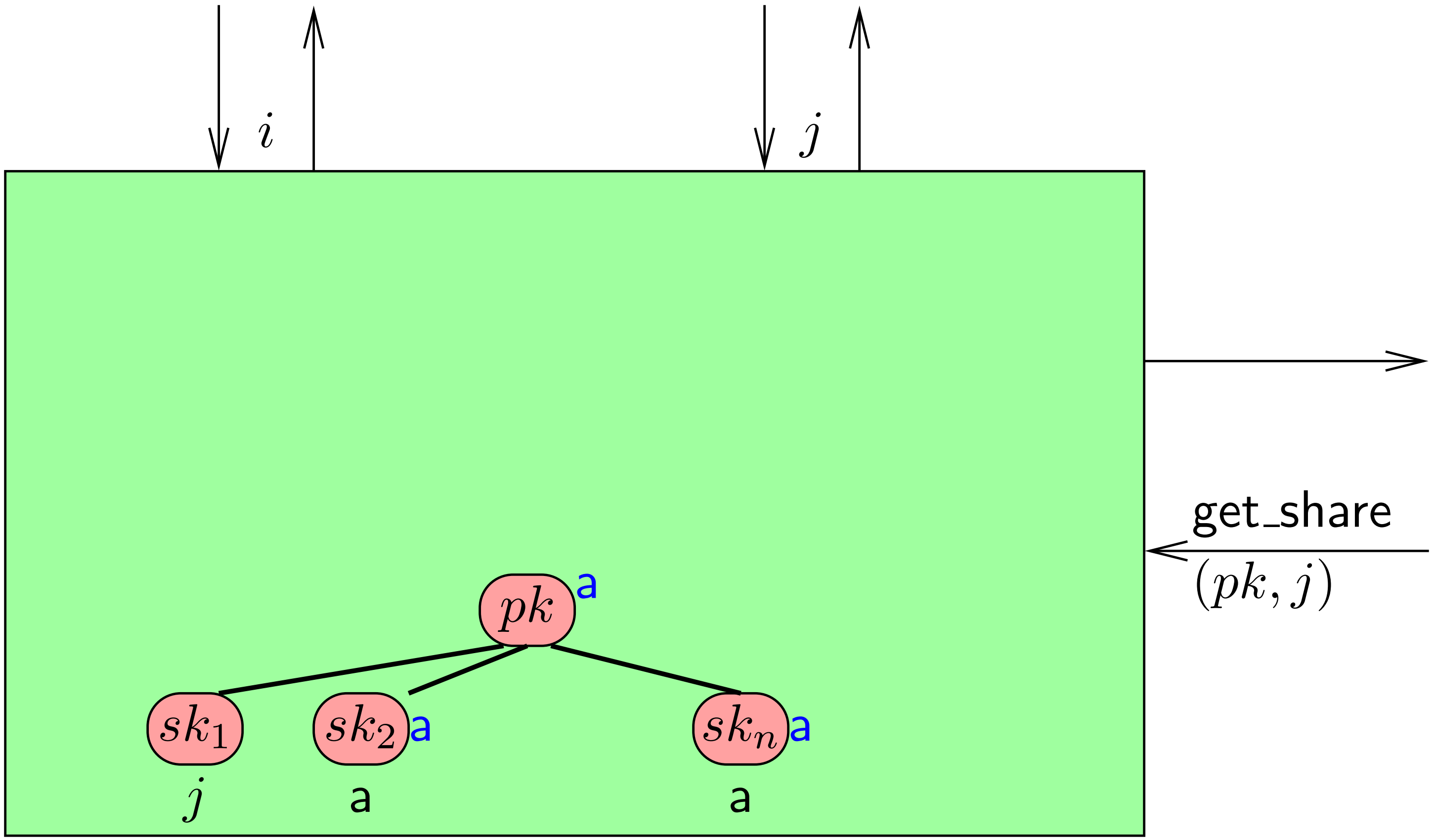
Abstract Library: key generation

- (Start to) generate a new set of keys
  - Specify the recipient of each secret key share



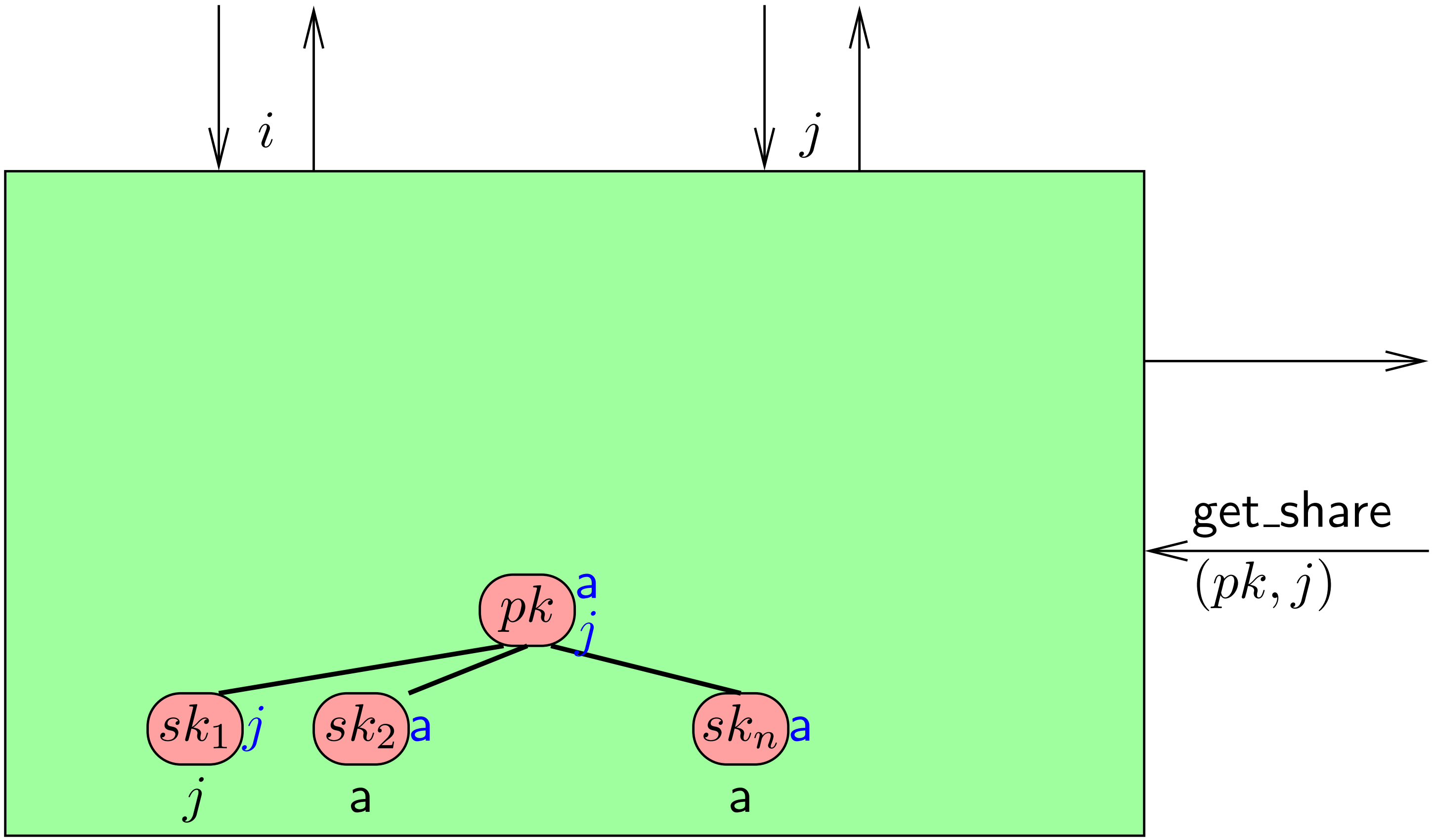
Abstract Library: key generation

- (Start to) generate a new set of keys
  - Specify the recipient of each secret key share



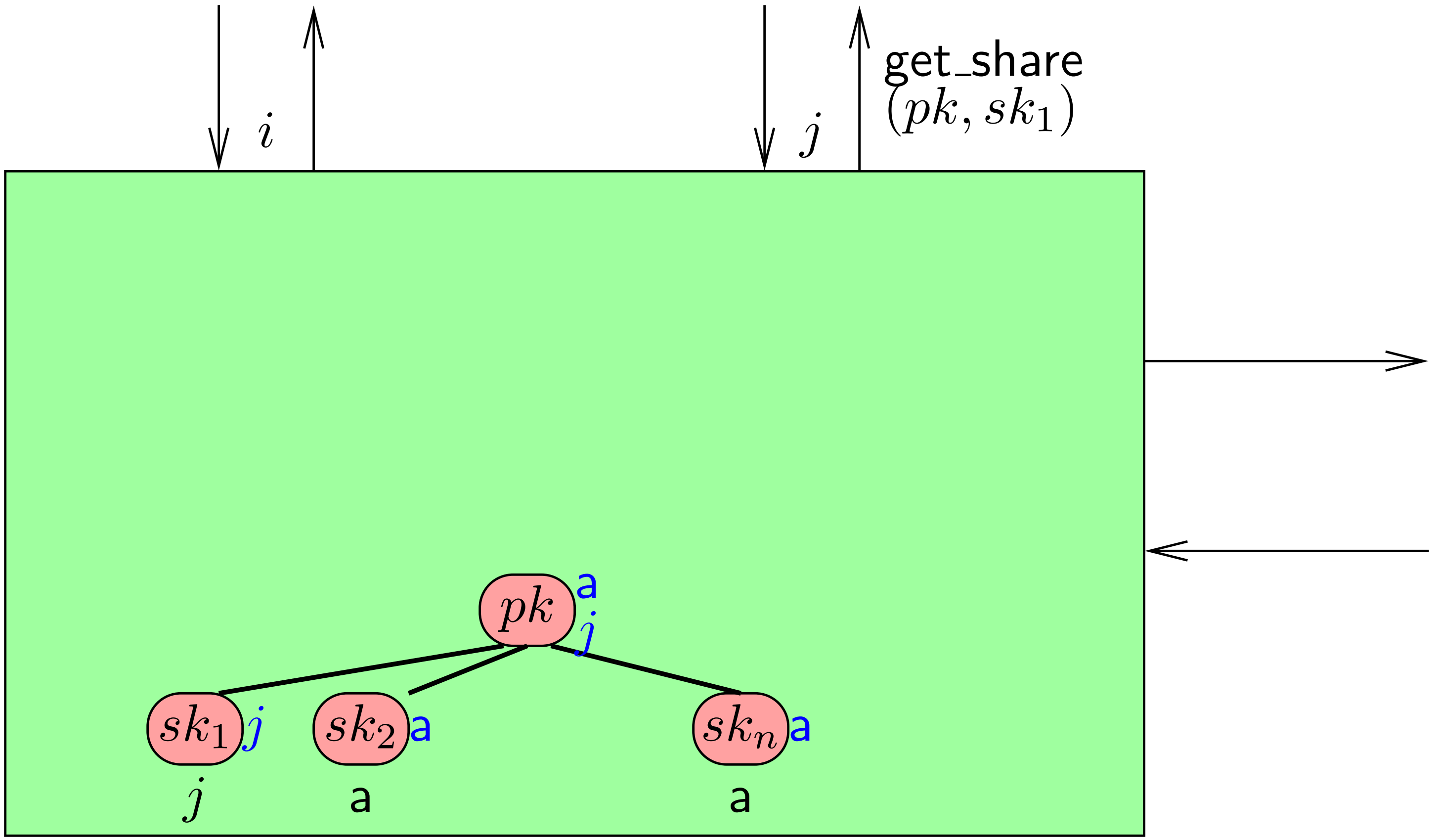
Abstract Library: key generation

- (Start to) generate a new set of keys
  - Specify the recipient of each secret key share



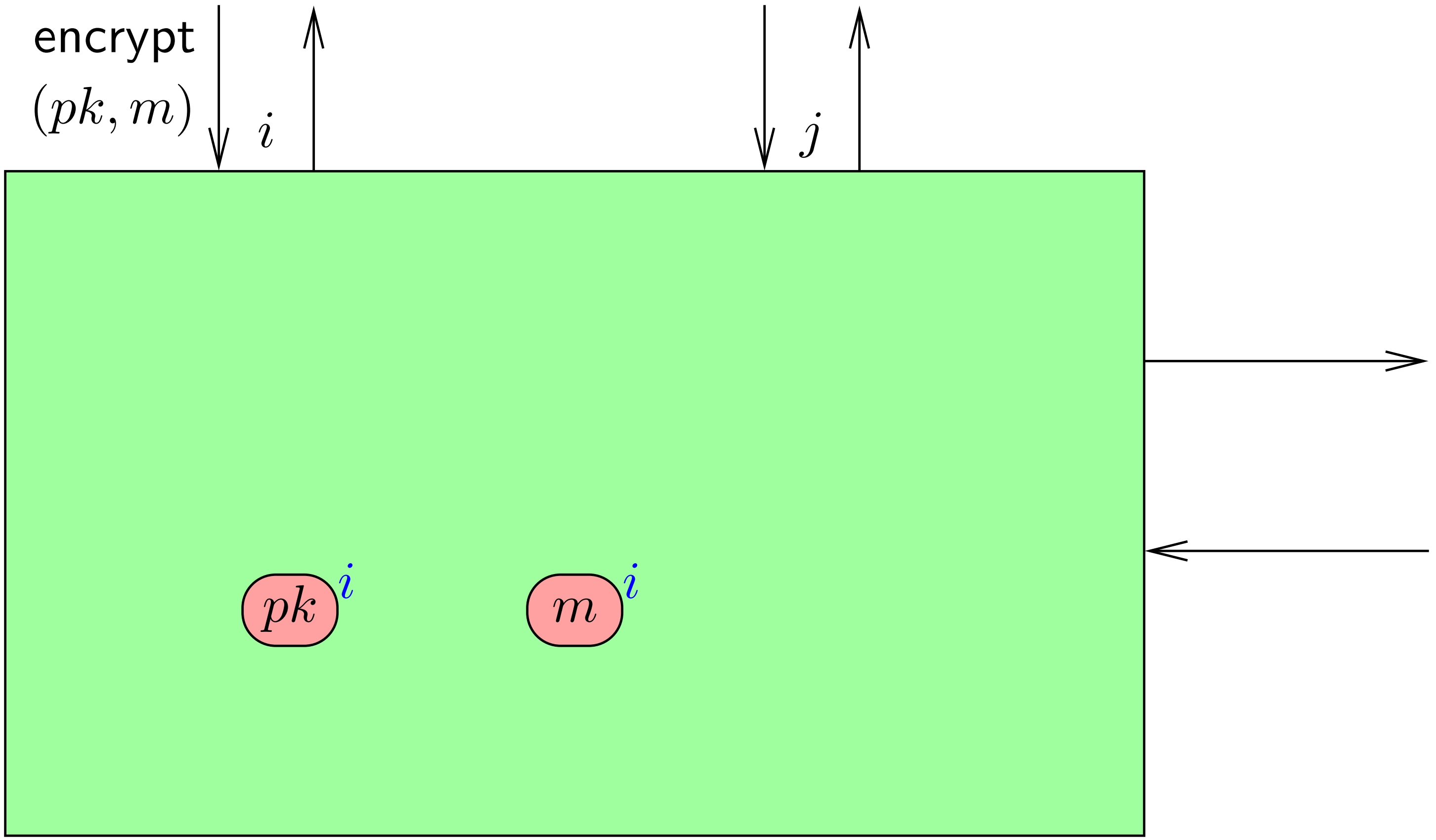
Abstract Library: key generation

- (Start to) generate a new set of keys
  - Specify the recipient of each secret key share



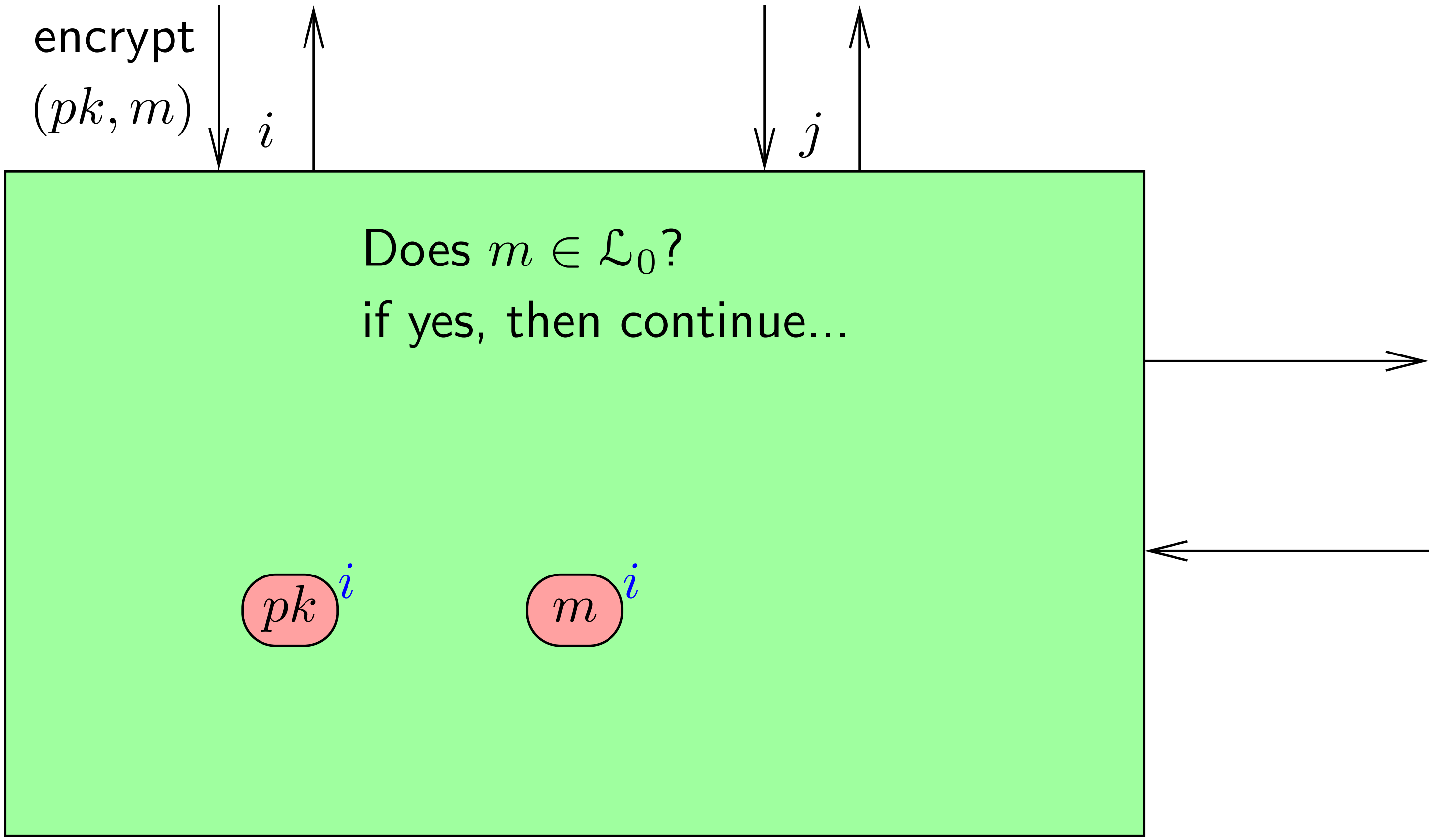
Abstract library: encryption

- Give the handles to the public key and the message.
  - Message must be a payload belonging to some $\mathcal{L}_0 \subseteq \{0,1\}^*$.



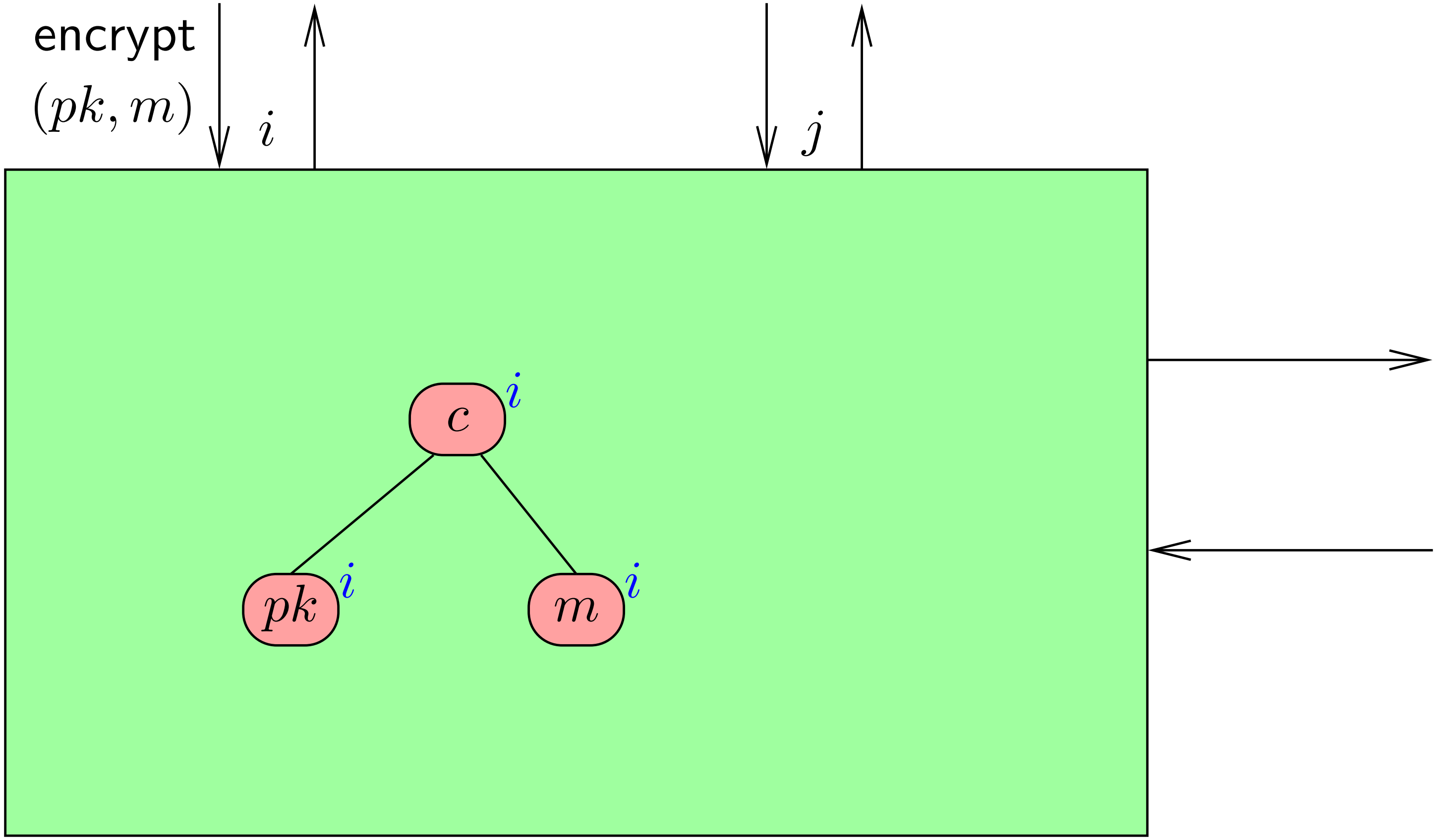
Abstract library: encryption

- Give the handles to the public key and the message.
  - Message must be a payload belonging to some $\mathfrak{L}_0 \subseteq \{0,1\}^*$.



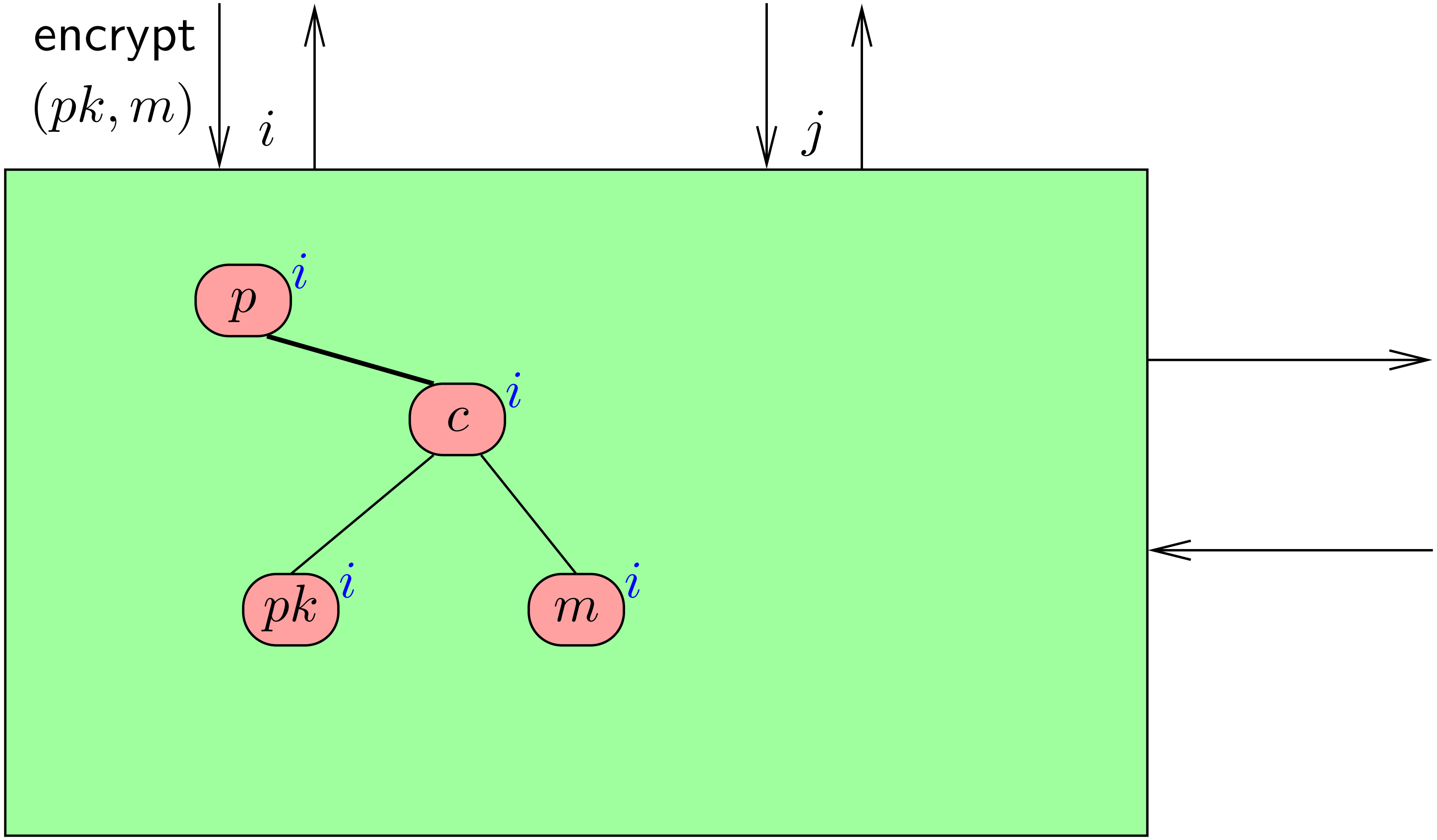
Abstract library: encryption

- Give the handles to the public key and the message.
  - Message must be a payload belonging to some $\mathcal{L}_0 \subseteq \{0,1\}^*$.



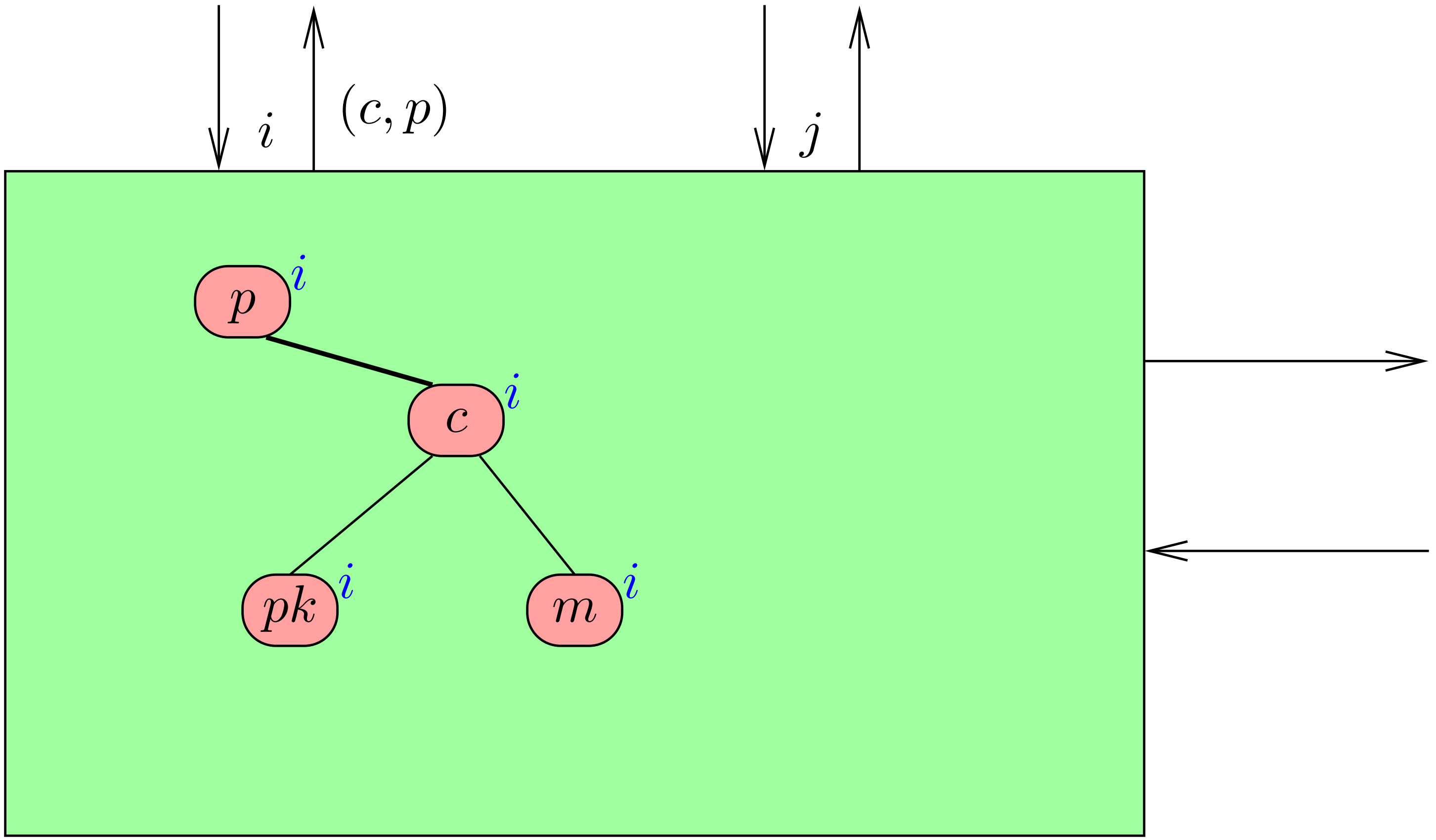
Abstract library: encryption

- Give the handles to the public key and the message.
  - Message must be a payload belonging to some $\mathcal{L}_0 \subseteq \{0,1\}^*$.



Abstract library: encryption

- Give the handles to the public key and the message.
  - Message must be a payload belonging to some $\mathfrak{L}_0 \subseteq \{0,1\}^*$.



Abstract library: decryption

- Given the handles to
  - Secret key share sk_i;
  - Ciphertexts $c_1, \ldots, c_k$;
    - (plaintexts: $m_1, \ldots, m_k$)
  - Validity proofs $p_1, \ldots, p_k$.
- The library will
  - Check the validity proofs.
    - c_i and p_i must be connected.
  - Construct a new payload term corresponding to $m_1 + \cdots + m_k$.
  - Construct new terms for a j-th plaintext share and its proof of validity.
    - point to pk, $c_1, \ldots, c_k$, $m_1 + \cdots + m_k$
  - Send back the handles for these last two terms.

Abstract library: empty validity proof

- Earlier, the adversary may have constructed a validity proof p without corresponding c.
- If $p = p_i$, then library sends find_witness(c_i, p_i) to the adversary.
- Adversary must respond with found_witness(c_i, p_i, m_i), where m_i is the plaintext of c_i.

Abstract library: empty validity proof

- Earlier, the adversary may have constructed a validity proof p without corresponding c.
- If $p = p_i$, then library sends find_witness(c_i, p_i) to the adversary.
- Adversary must respond with found_witness(c_i, p_i, m_i), where m_i is the plaintext of c_i.
- To find m_i, the adversary is allowed to parse terms and store new payloads in the abstract library.
- The adversary is not allowed to communicate with anyone else.

Abstract library: combining plaintext shares

- Given the handles to
  - Public key pk;
  - Plaintext shares $ds_{i_1}, \dots, ds_{i_t}$;
  - Their validity proofs $dp_{i_1}, \dots, dp_{i_t}$.
- The library will
  - Check that the shares come from the same set of ciphertexts, created with the public key pk;
  - Check the validity proofs;*
  - Return the handle to the plaintext referenced by all $ds_\star$.

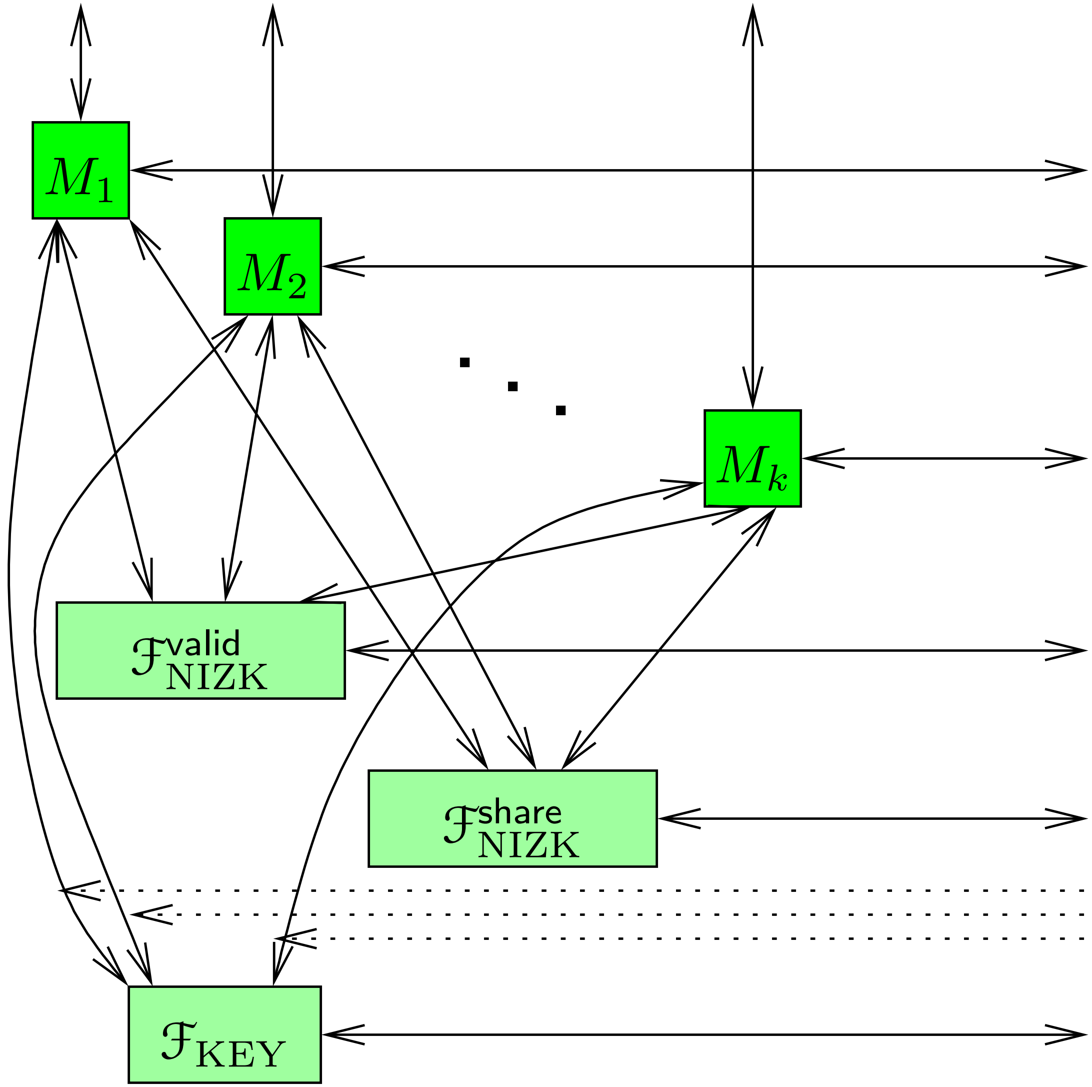
Abstract library: adversarial commands

- Create a new public key
  - Only pk, not $sk_1, \ldots, sk_n$
- Create an invalid (empty) encryption / validity proof
- Decrypt without checking validity proofs
- Combine without checking validity proofs
- Create an invalid plaintext share or validity proof
- Transform a validity proof of a plaintext share
- Parse terms
  - Except for ciphertexts (only gets the length of plaintext)

Combining ptxt shares: invalid public key

- Given the handles to
  - Public key pk, created by the adversary;
  - Plaintext shares $ds_{i_1}, \ldots, ds_{i_t}$;
  - Their validity proofs $dp_{i_1}, \ldots, dp_{i_t}$.
- The library will
  - Check that the shares come from the same set of ciphertexts, created with the public key pk;
  - Forward the combine-command to the adversary
    - Translate the handles
  - Receive a handle to the payload
  - Forward it to the user

Real library: structure


M_1
M_2
M_k
$\mathcal{F}^{\text{valid}}_{\text{NIZK}}$
$\mathcal{F}^{\text{share}}_{\text{NIZK}}$
$\mathcal{F}_{\text{KEY}}$

Source of components

- $\mathcal{F}_{\mathrm{NIZK}}$
  - Jens Groth, Rafail Ostrovsky, Amit Sahai. Perfect Non-Interactive Zero-Knowledge for NP. EUROCRYPT 2006.
- $\mathcal{F}_{\mathrm{KEY}}$
  - Douglas Wikström. Universally Composable DKG with Linear Number of Exponentiations. SCN 2004.
- Threshold homomorphic encryption
  - Ivan Damgård, Mads Jurik. A generalisation, a simplification and some applications of Paillier's probabilistic public-key system. PKC 2001.

Conclusions

- A good abstraction significantly simplifies the analysis of protocols.
- The monolithic library can offer significantly higher abstractions than stand-alone abstract functionalities.

Conclusions

- A good abstraction significantly simplifies the analysis of protocols.
- The monolithic library can offer significantly higher abstractions than stand-alone abstract functionalities.
- Future work:
  - improve the combination possibilites of ciphertexts.
  - consider other primitives, like secret sharing.