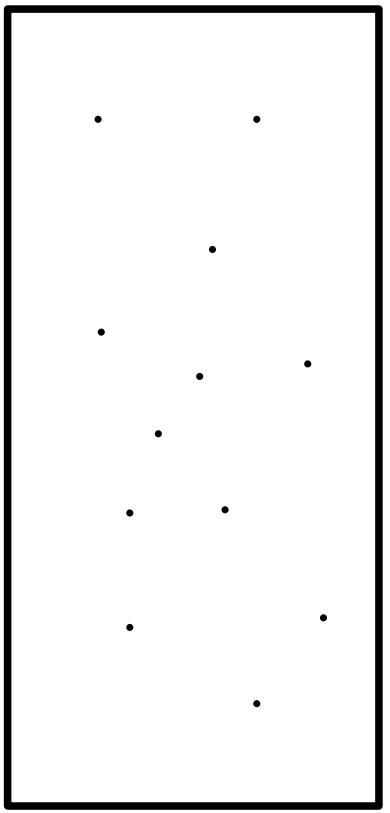
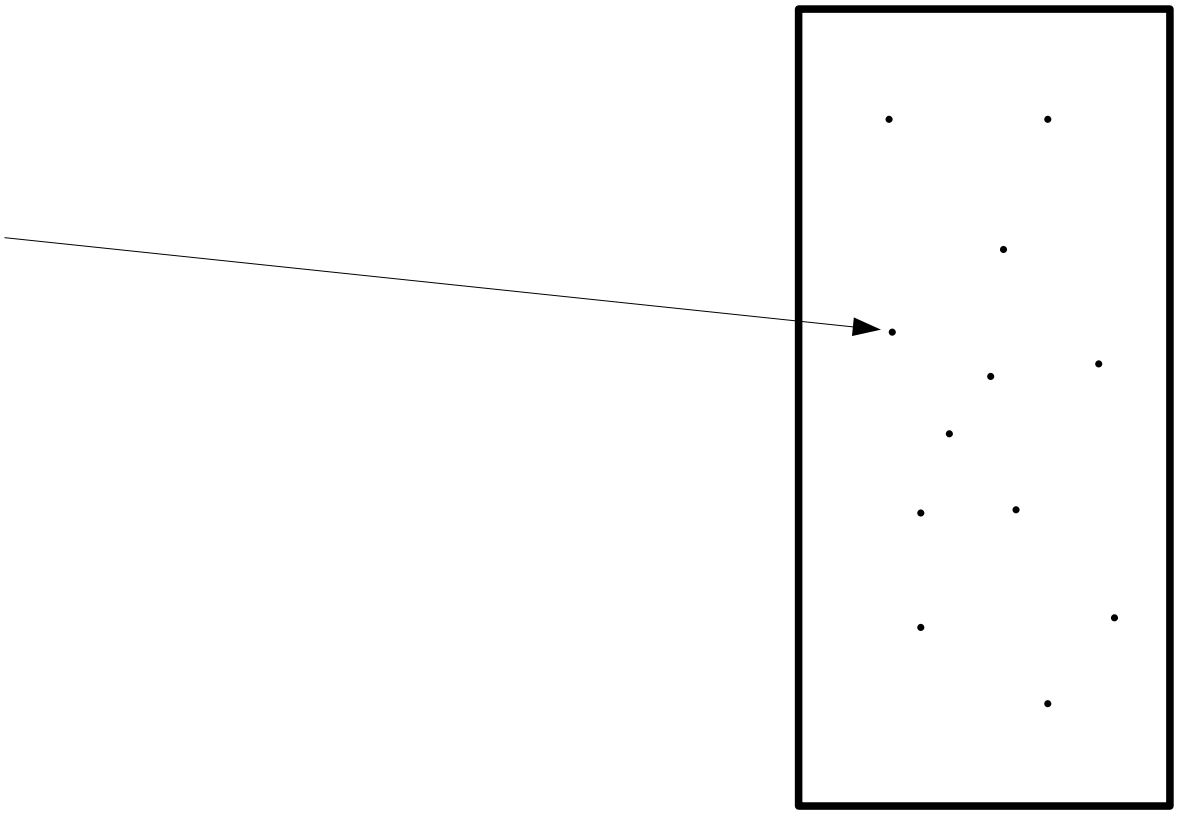
Hashing (Message Digest)

Hashing (Message Digest)

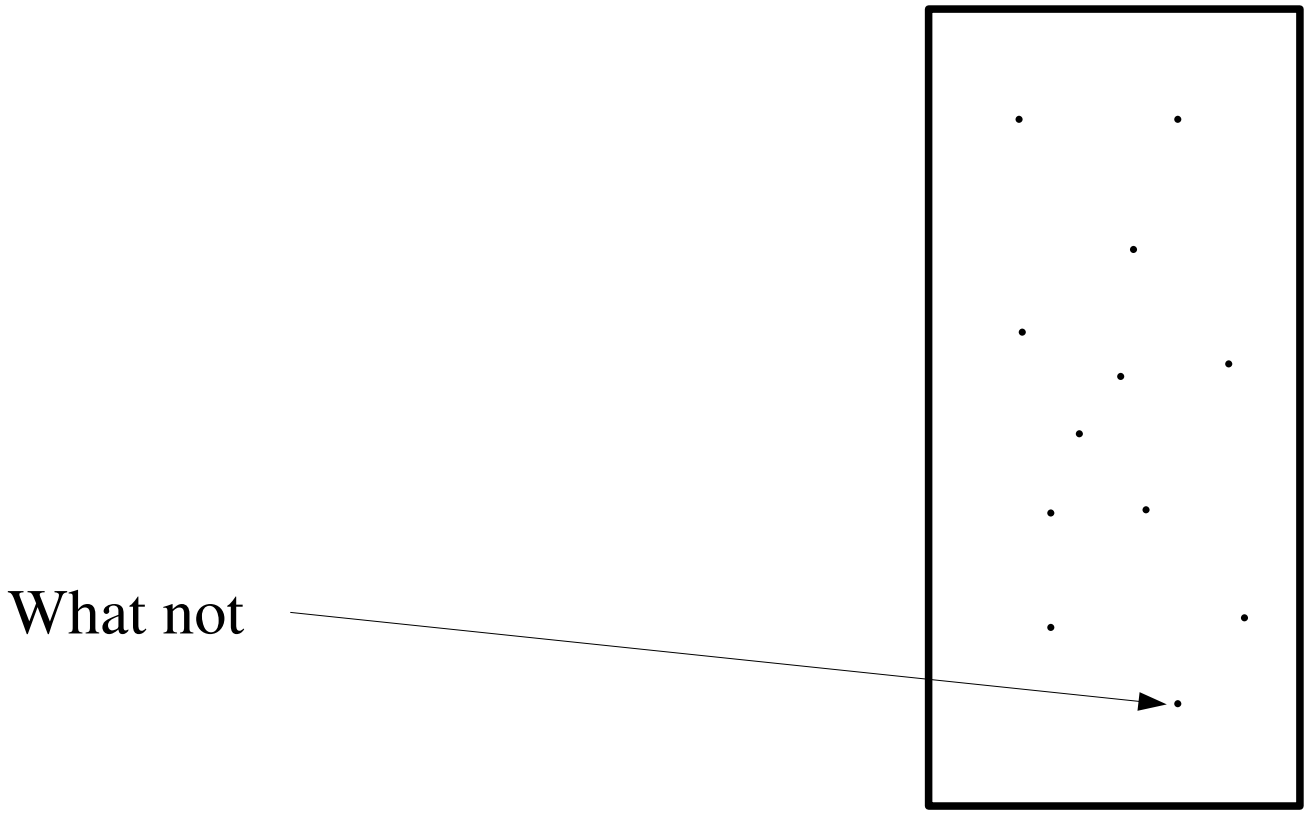


Hashing (Message Digest)

Hello There



Hashing (Message Digest)



Hash Function – One way

Needed properties for cryptographically secure hash:

1. Computationally infeasible to find the message that has given the hash

2. Should be highly improbable for two messages to hash to the same number (and to find two messages with the same hash).

   Message substitution computationally feasible otherwise

Hash Function – One way

Why worry about two messages with same hashes?

Application: integrity check, message substitution is possible

Example: possible to find two different messages that have opposite meanings. AA gets assigned to write an email firing employee that AA likes. Email will have hash integrity check. It will be checked by boss and then signed before being sent.

I am writing {this memo} to {demand | request | inform you} that {Fred | Mr. Jones} {must} be {fired | terminated} {at once | immediately}. As the {July 11 | 11 July} {memo | memorandum} {from | issued by} {personnel | HR | Human Relations} states, to meet {our | the corporate} {quarterly | third quarter} budget {target | goal}, {we must..

I am writing {this letter | this memo | this memorandum | } to {officially} commend {Fred | Mr. Jones} for this {courage and independent thinking | independent thinking and courage}. {He | Fred | Mr. Jones} {clearly | } understands {the need | how} to get {the | his} job {done | accomplished} {at all costs | by whatever means necessary}, and {knows ...

Hash Function – One way

Appearance to a cracker:

1. Looking at output, any bit should be 1 about ½ the time 0010111...1...001110

2. Each output should have about ½ of its bits set to 1

3. Any two outputs should be uncorrelated no matter how similar the inputs are

Hash Function – One way

Birthday Problem:

Assume a hash function *H* that pretty much randomly maps an integer input to an integer output. Suppose the number of output values for *H* is *k*. Pick *n* input integers randomly. How large should *n* be so that the probability that at least one pair of input integers map to the same output is 1/2?

Answer:

Pr(some pair of inputs map to the same number) > ½ if

$n > \sqrt{2k}$

For $k=365$ days, $n = 27$

Hash Function – One way

Birthday Problem:

Assume a hash function H that pretty much randomly maps an integer input to an integer output. Suppose the number of output values for H is k. Pick n input integers randomly. How large should n be so that the probability that at least one pair of input integers map to the same output is 1/2?

Answer:

Pr(some pair of inputs map to the same number) > ½ if

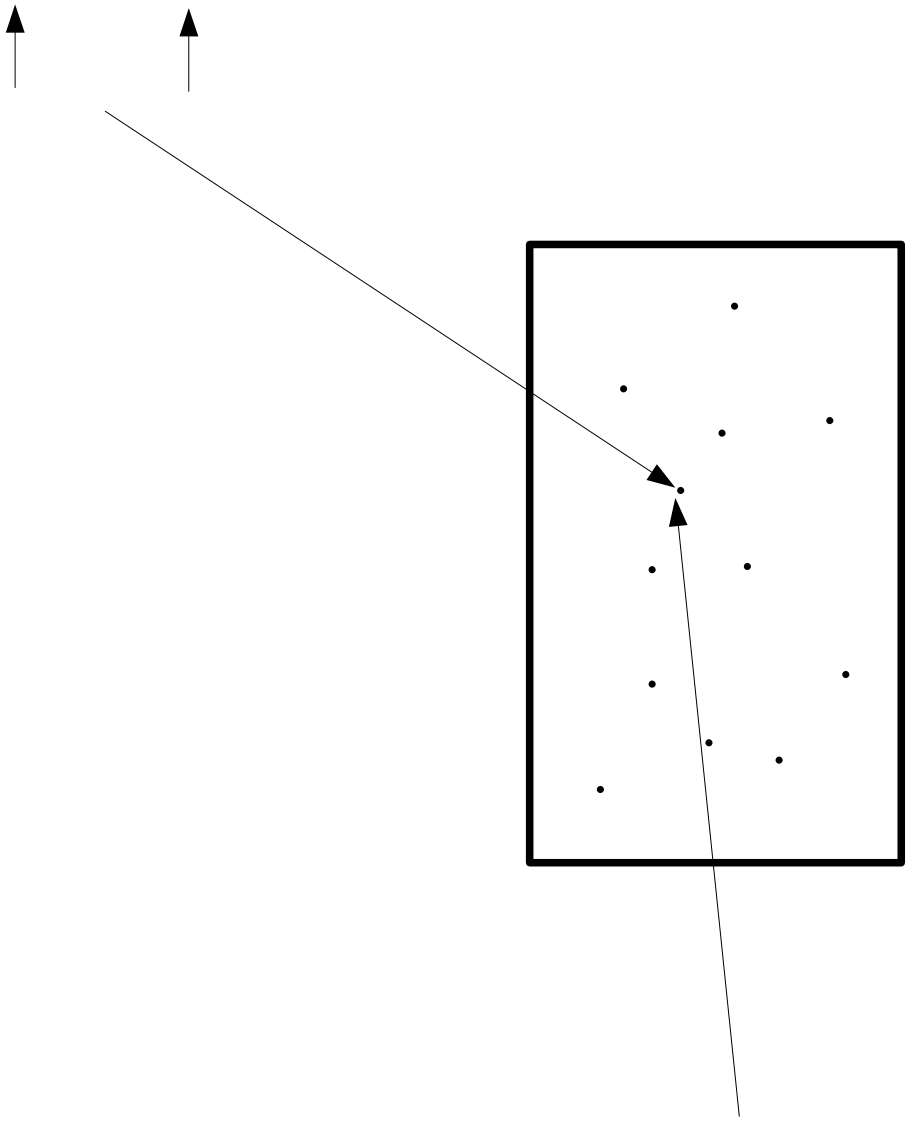
$n > \sqrt{2k}$

For $k=365$ days, $n = 27$

SHA: 160 bits output hence 2^{80} numbers will likely produce a pair that has a collision

Hashing (Message Digest)

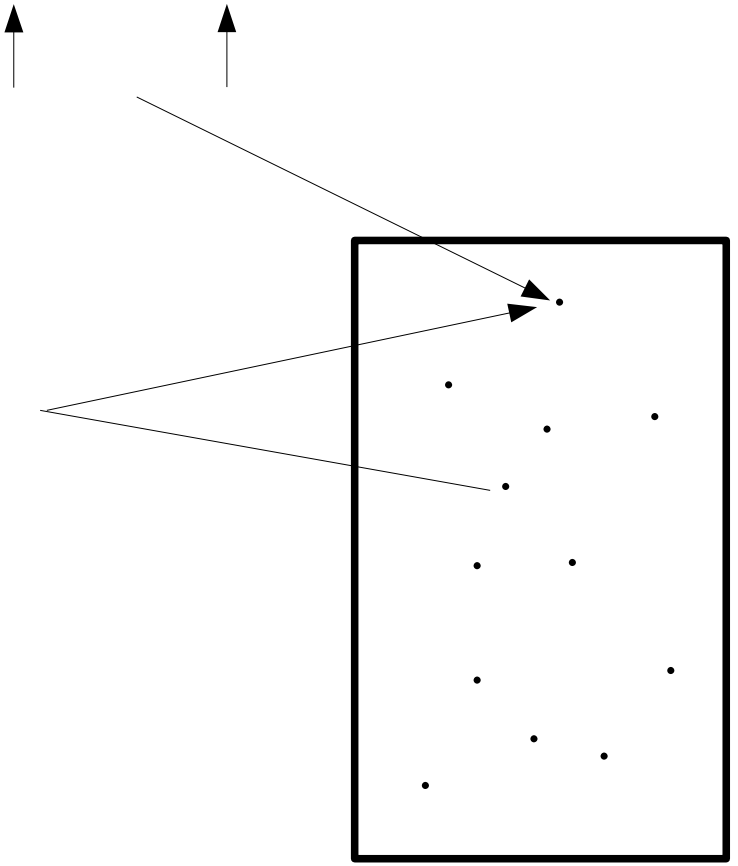
The little brown fox jumped over the lazy dog's back



Secret

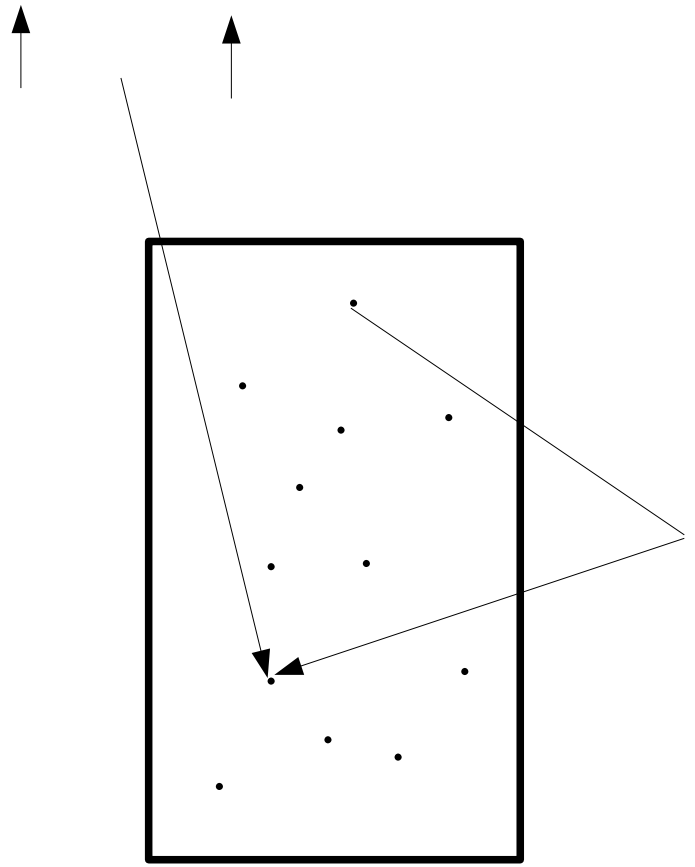
Hashing (Message Digest)

The little brown fox jumped over the lazy dog's back



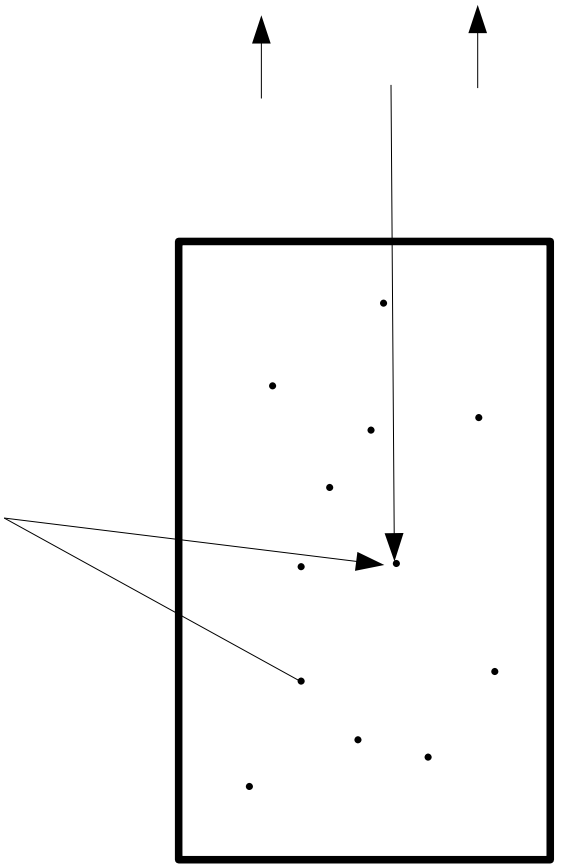
Hashing (Message Digest)

The little brown fox jumped over the lazy dog's back



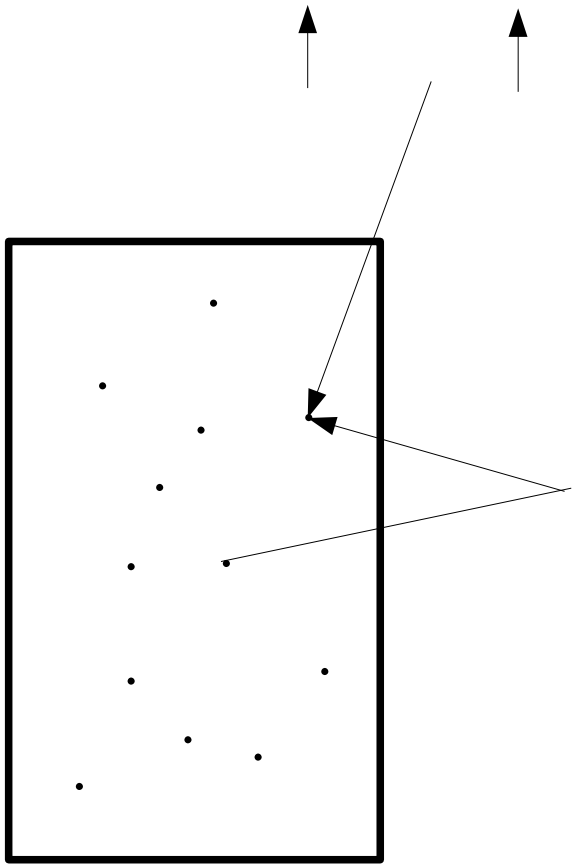
Hashing (Message Digest)

The little brown fox jumped over the lazy dog's back



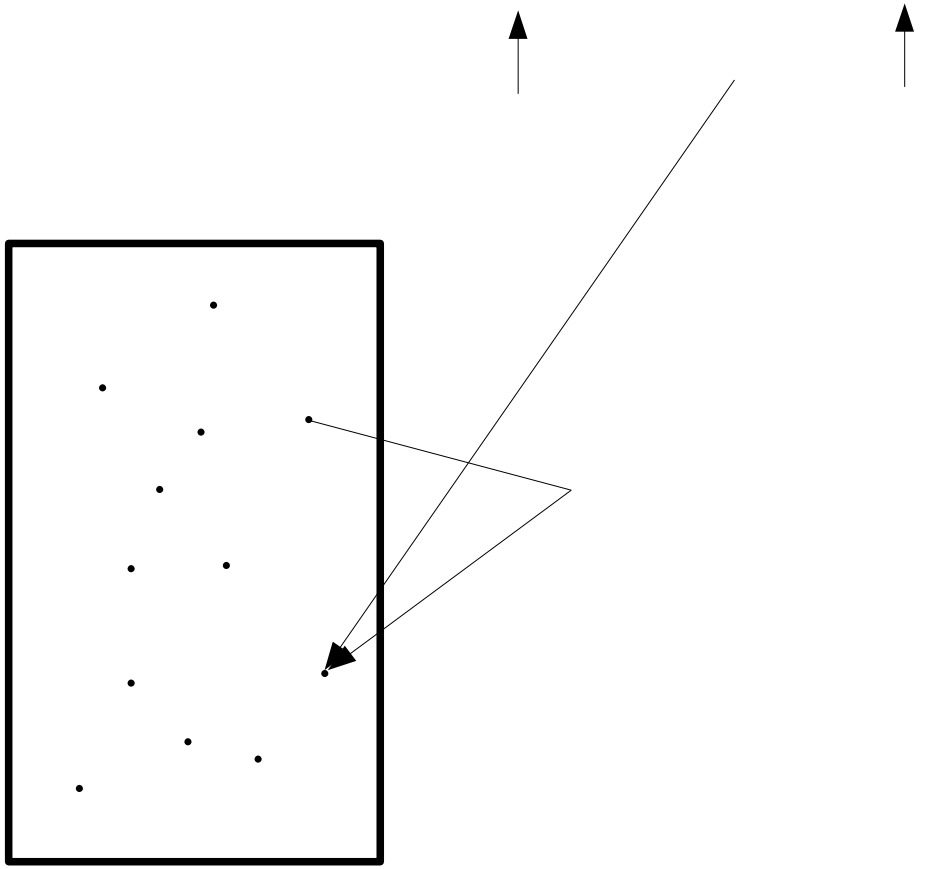
Hashing (Message Digest)

The little brown fox jumped over the lazy dog's back



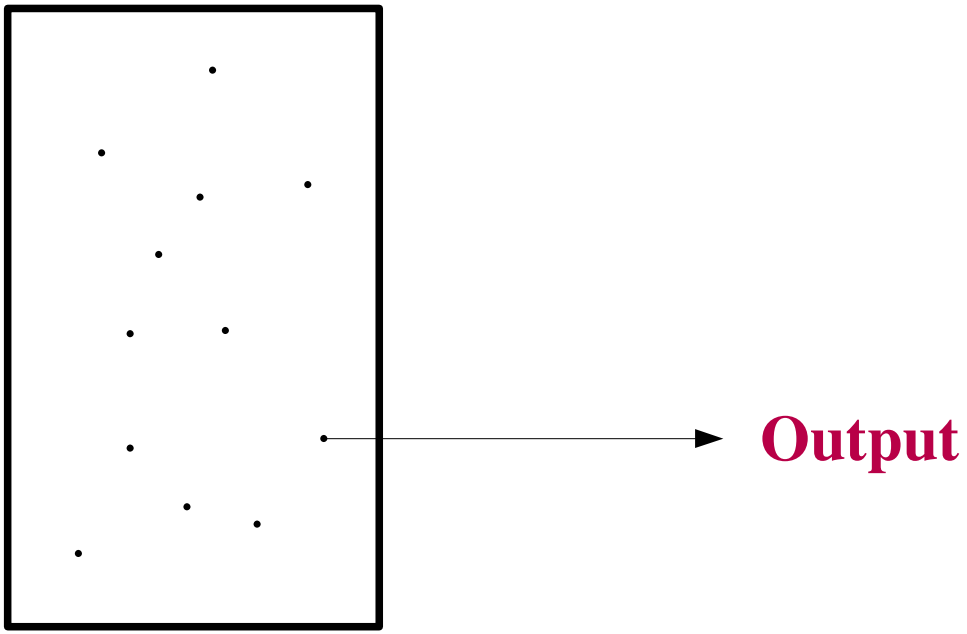
Hashing (Message Digest)

The little brown fox jumped over the lazy dog's back



Hashing (Message Digest)

The little brown fox jumped over the lazy dog's back



Hashing (Message Digest)

Java:

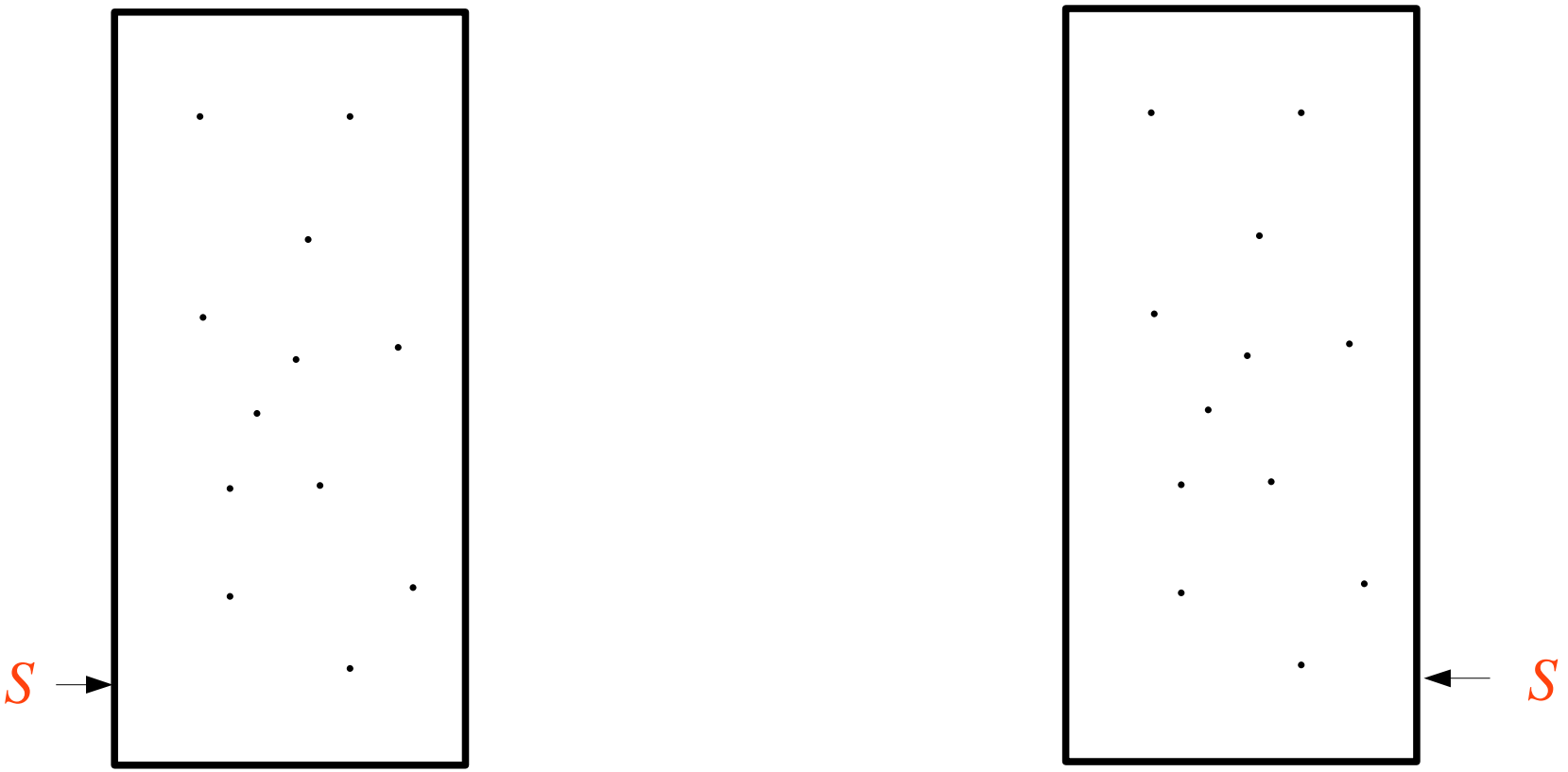
class SecureRandom – produces a cryptographically secure random number

class MessageDigest – produces a hash of a number of bytes depending on the algorithm used (SHA, MD5, etc.)

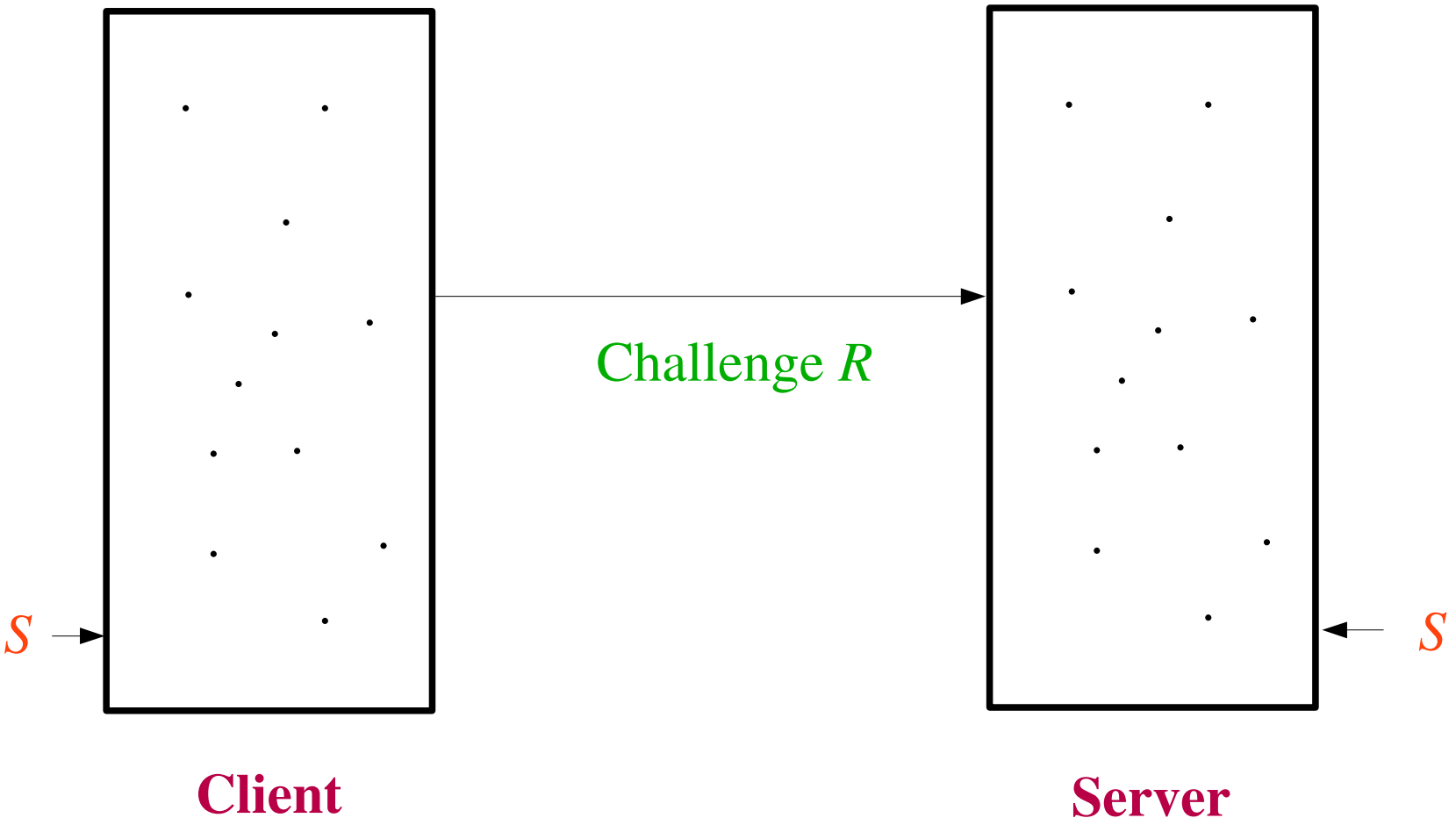
Cryptographically secure random number:

1. Given the first k bits of a CSPRN there is no efficient algorithm that will predict with probability better than 1/2 what the next bit will be
2. If information about a state of the computation is revealed at some point, it should be infeasible to produce the output bits that has been produced up to that point. If someone indexes into a published random sequence of bits – this point is violated.

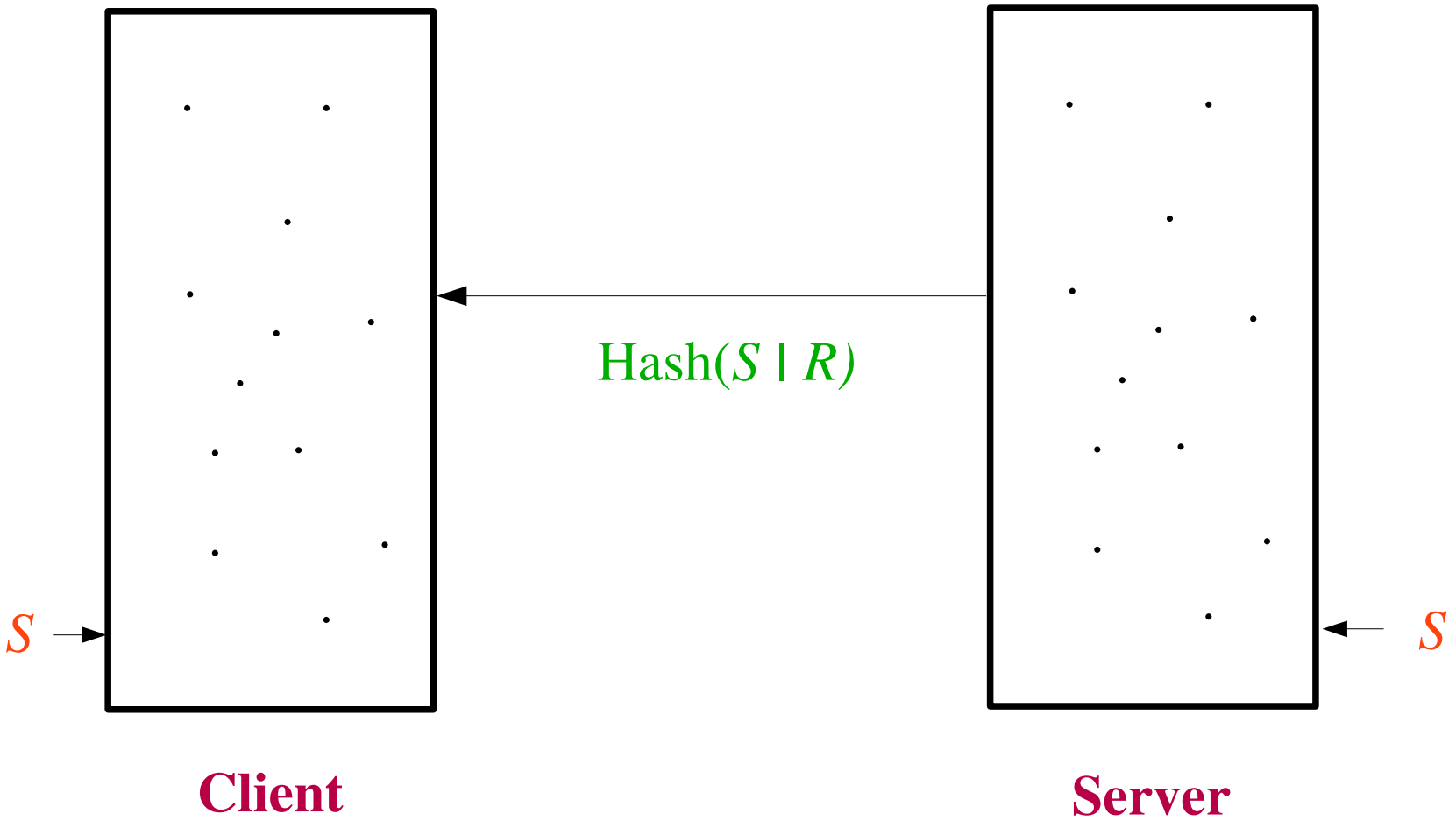
Hashing - Authentication



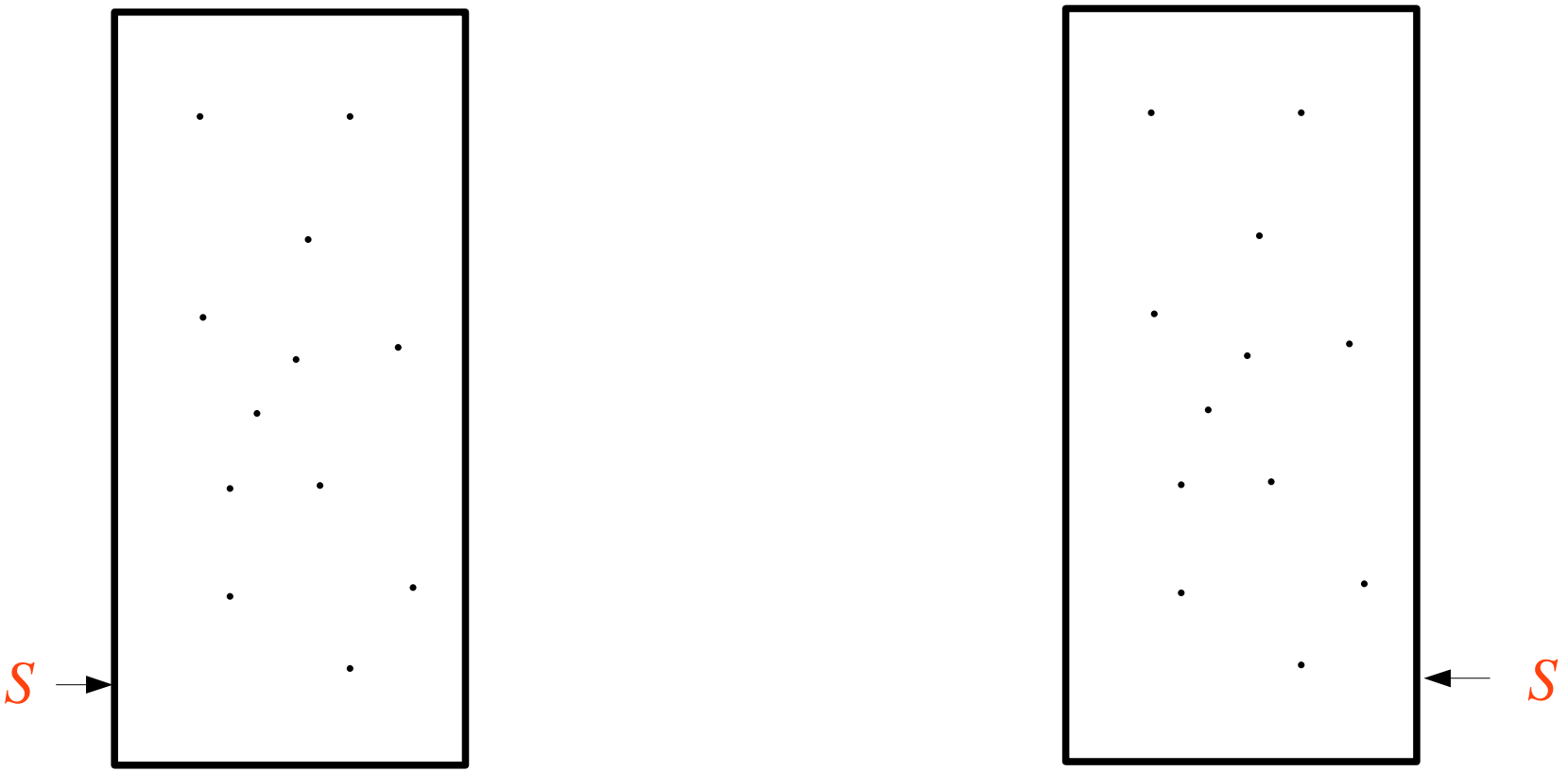
Hashing - Authentication



Hashing - Authentication



Hashing - Authentication

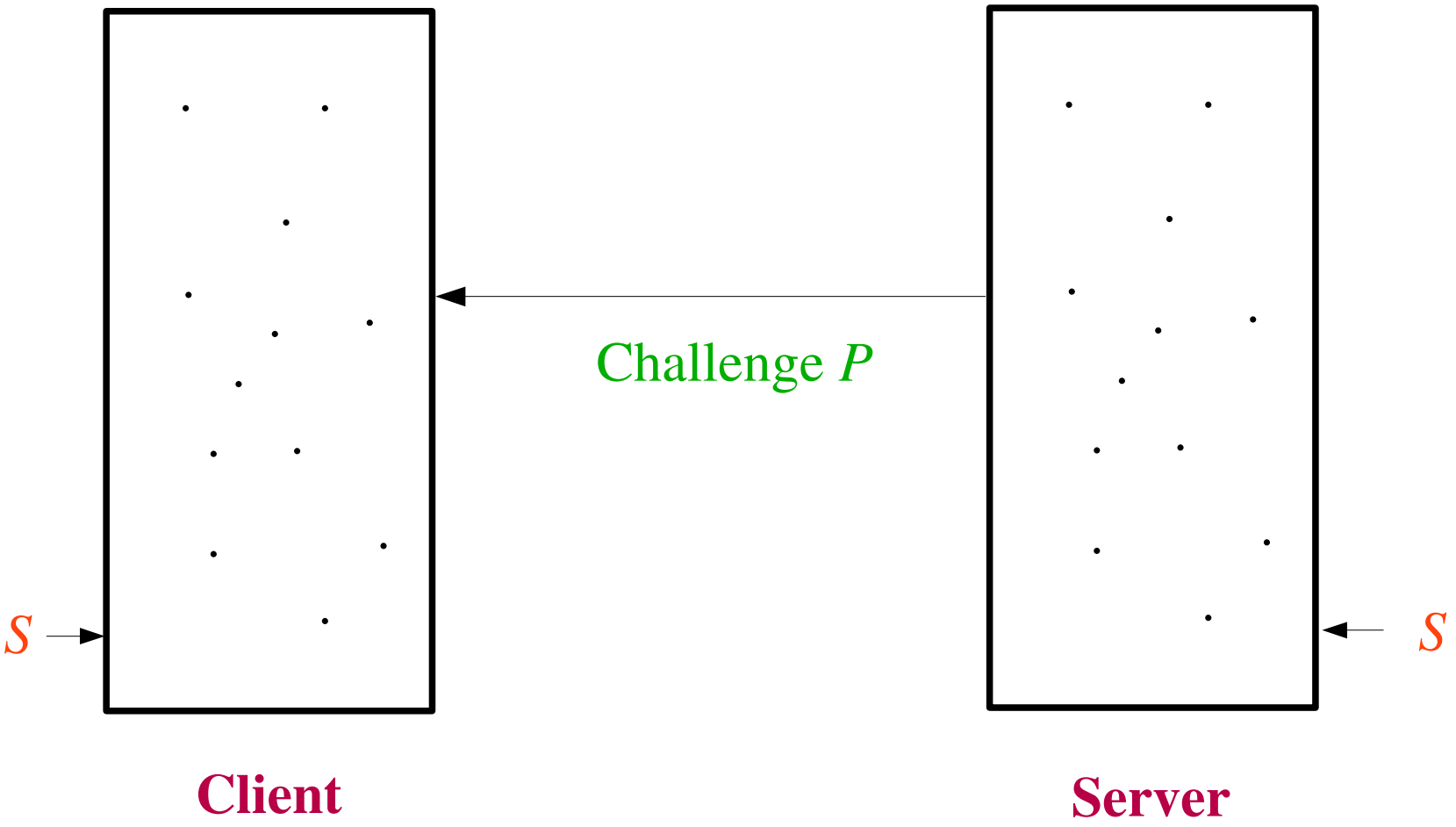


Client

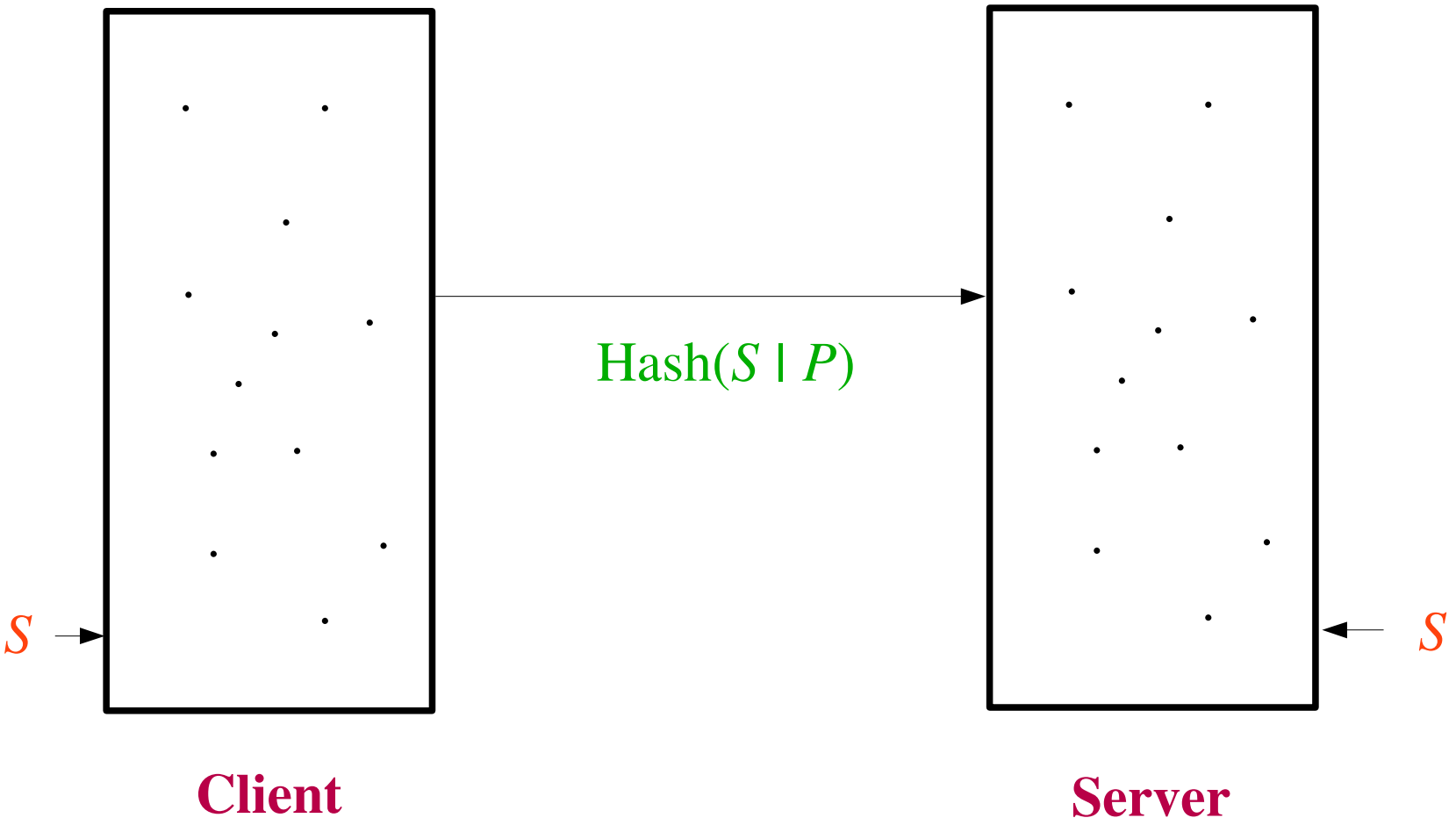
Check Hash($S \mid R$)?

Server

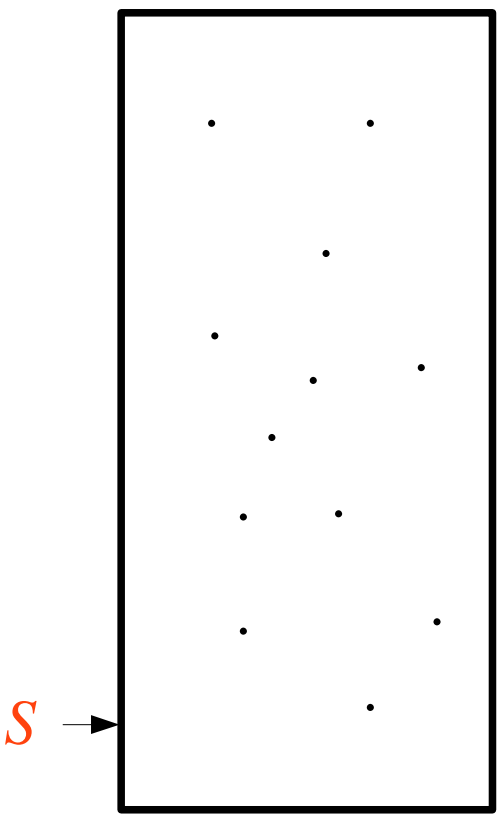
Hashing - Authentication



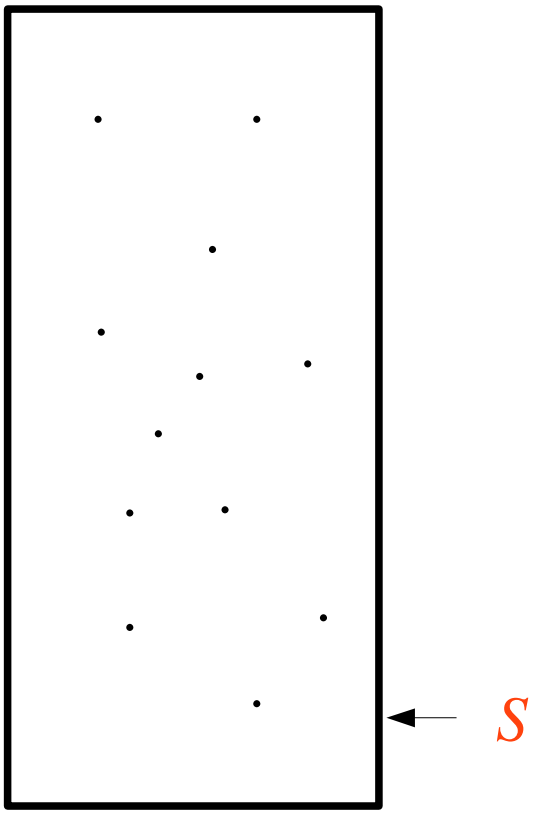
Hashing - Authentication



Hashing - Authentication



Client



Server

Check Hash(S | P)?

Hashing - Message Integrity

Try same as authentication except concatenate the message with the secret and pad (send Hash($S \mid M$)).

Hashing - Message Integrity

Try same as authentication except concatenate the message with the secret and pad (send Hash($S \mid M$)). Unfortunately, attacker can append a message since s/he knows Hash($S \mid M$) and the Hash algorithm.

Hashing - Message Integrity

Try same as authentication except concatenate the message with the secret and pad (send Hash($S \mid M$)). Unfortunately, attacker can append a message since s/he knows Hash($S \mid M$) and the Hash algorithm.

Try Hash($M \mid S$). But then if the digest for two messages is the same, the MAC for both messages is the same – doesn't smell right.

Hashing - Message Integrity

Try same as authentication except concatenate the message with the secret and pad (send Hash($S \mid M$)). Unfortunately, attacker can append a message since s/he knows Hash($S \mid M$) and the Hash algorithm.

Try Hash($M \mid S$). But then if the digest for two messages is the same, the MAC for both messages is the same – doesn't smell right.

Try using only ½ of the bits of the digest – then attacker cannot append a message so easily – but the digest should be longer to compensate.

Hashing - Message Integrity

Try same as authentication except concatenate the message with the secret and pad (send Hash($S \mid M$)). Unfortunately, attacker can append a message since s/he knows Hash($S \mid M$) and the Hash algorithm.

Try Hash($M \mid S$). But then if the digest for two messages is the same, the MAC for both messages is the same – doesn't smell right.

Try using only ½ of the bits of the digest – then attacker cannot append a message so easily – but the digest should be longer to compensate.

Try Hash($S \mid M \mid S$).

Hashing - Message Integrity

Try same as authentication except concatenate the message with the secret and pad (send Hash($S \mid M$)). Unfortunately, attacker can append a message since s/he knows Hash($S \mid M$) and the Hash algorithm.

Try Hash($M \mid S$). But then if the digest for two messages is the same, the MAC for both messages is the same – doesn't smell right.

Try using only ½ of the bits of the digest – then attacker cannot append a message so easily – but the digest should be longer to compensate.

Try Hash($S \mid M \mid S$).

HMAC, the winner:

1. Concatenate secret to front of message
2. Take the hash
3. Concatenate the secret to the front of the hash
4. Take the hash : **Hash($S_1 \mid$ Hash($S_2 \mid M$))**

   where S_1 and S_2 are derived from S

Hashing - Message Integrity

HMAC, the winner:

1. Concatenate secret to front of message
2. Take the hash
3. Concatenate the secret to the front of the hash
4. Take the hash : **Hash(S_1 | Hash(S_2 | M))**

$S_1 = S \oplus \text{constant}_1, \quad S_2 = S \oplus \text{constant}_2$

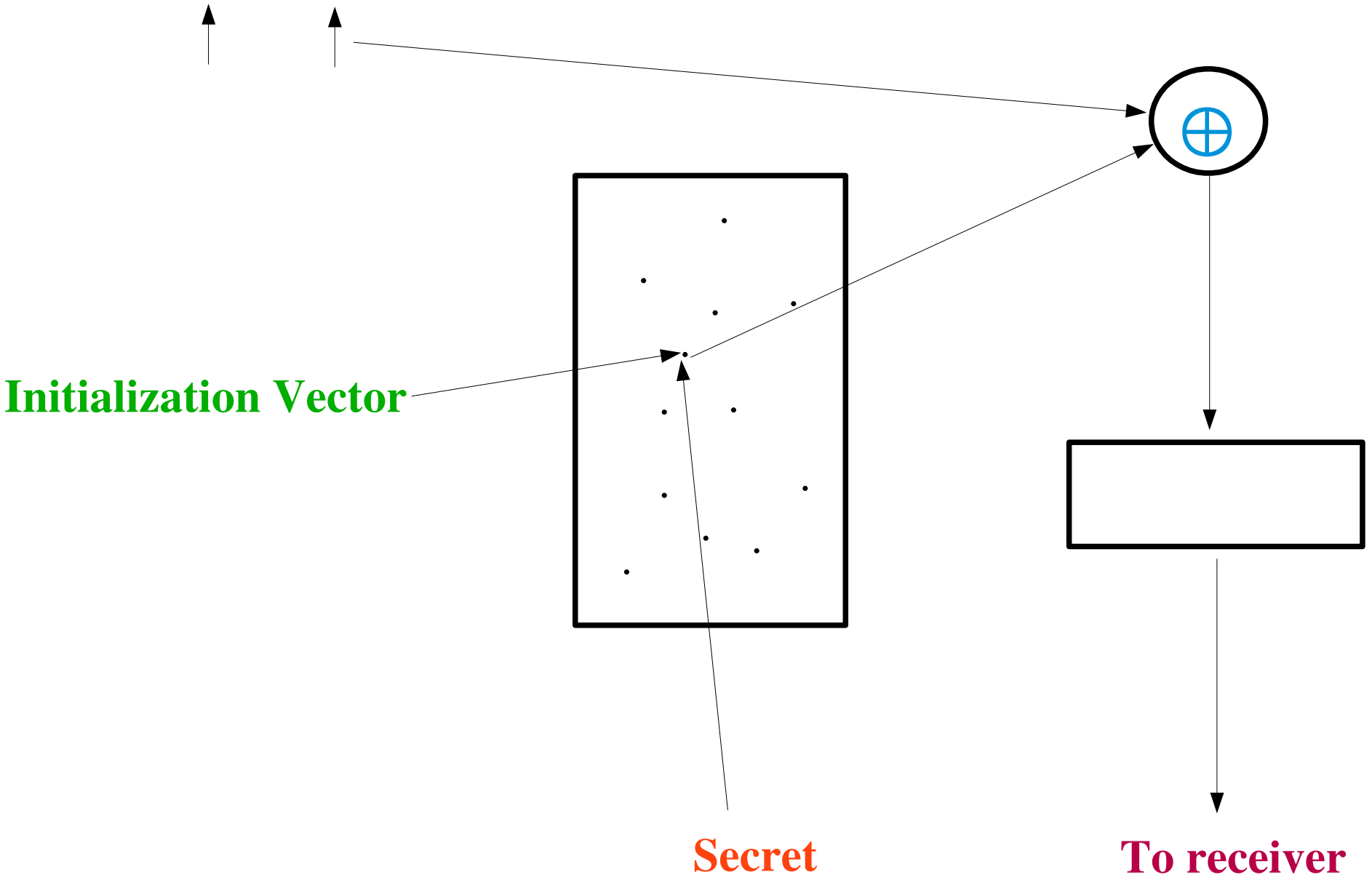
Reason: It has been shown that HMAC has the following properties:

1. It is infeasible to find two inputs that yield the same output
2. An attacker that does not know S cannot compute the proper digest involving S and M even if the attacker knows the digest involving S and M_i for any M_i such that M_i is not the same as M

Provided: the underlying hash function is secure (has the same properties)

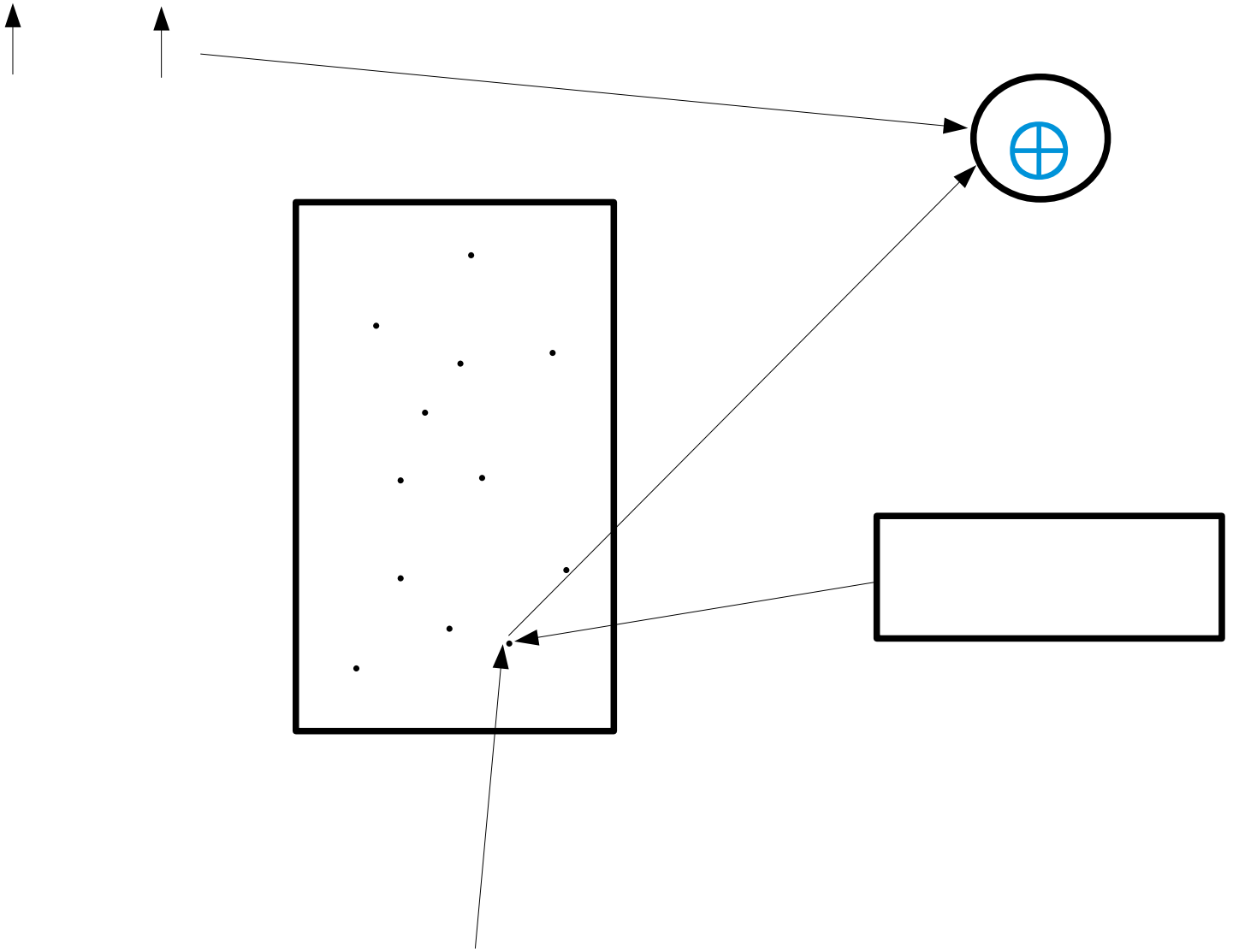
Hashing - Encryption

The little brown fox jumped over the lazy dog's back



Hashing - Encryption

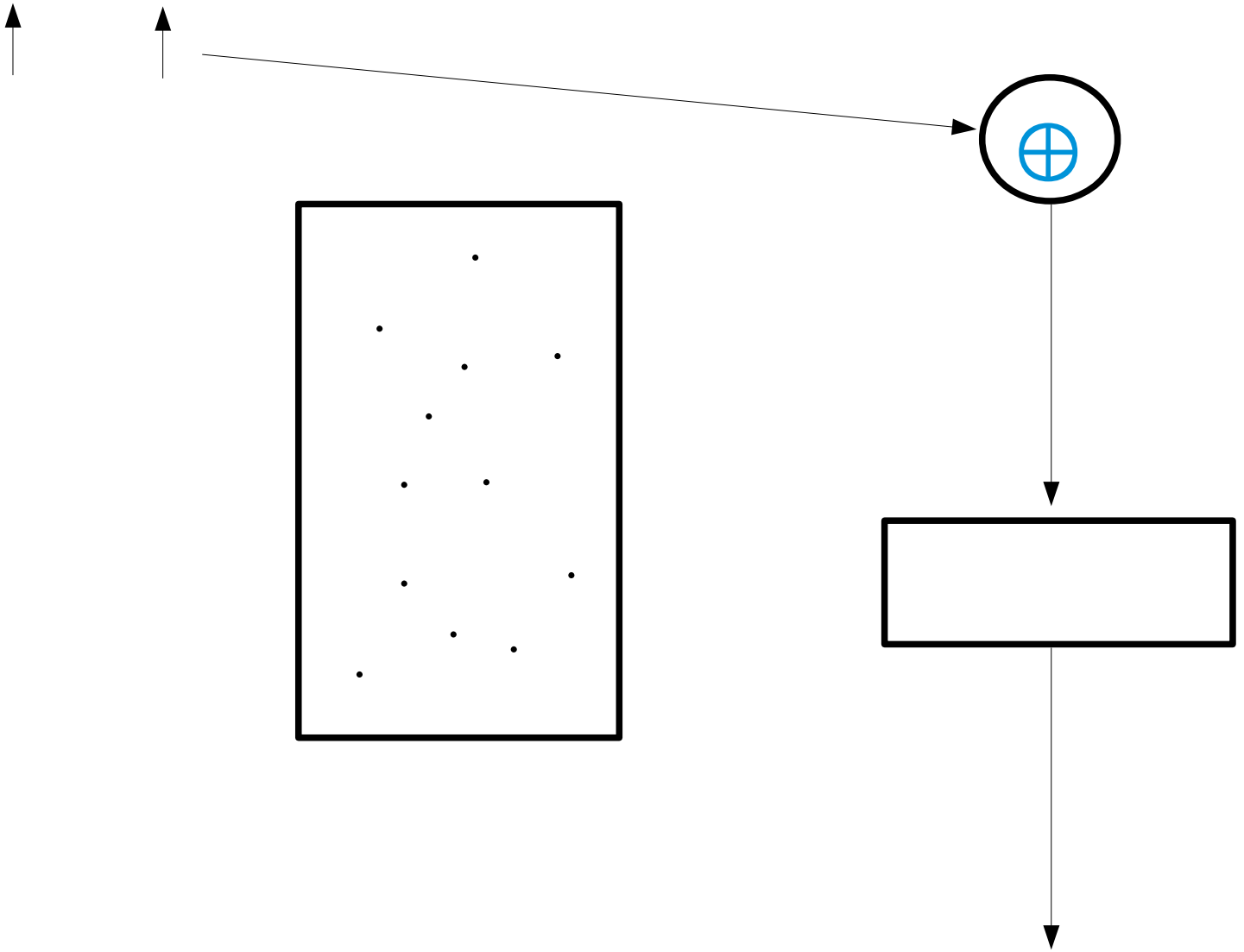
The little brown fox jumped over the lazy dog's back



Secret

Hashing - Encryption

The little brown fox jumped over the lazy dog's back

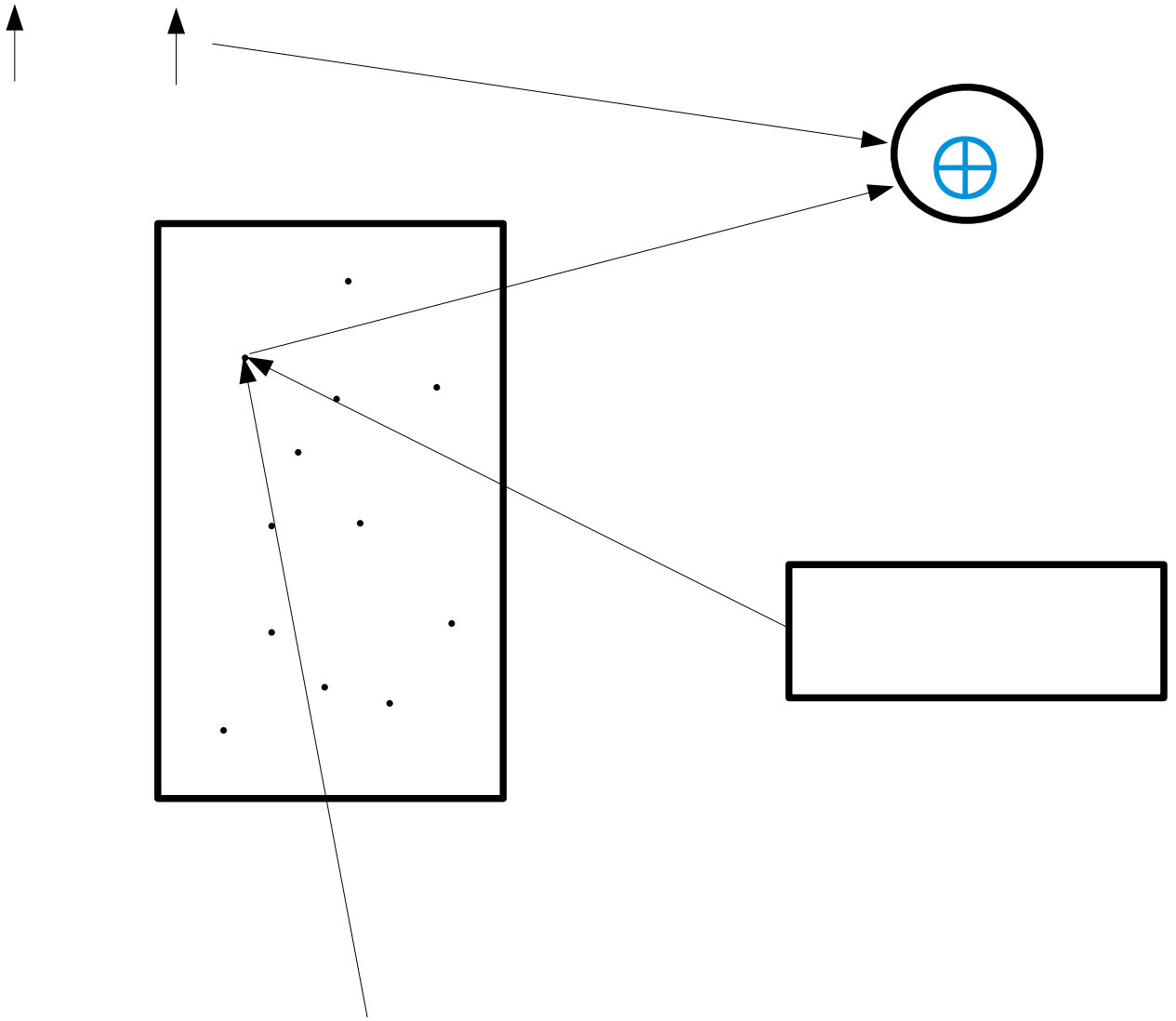


Secret

To receiver

Hashing - Encryption

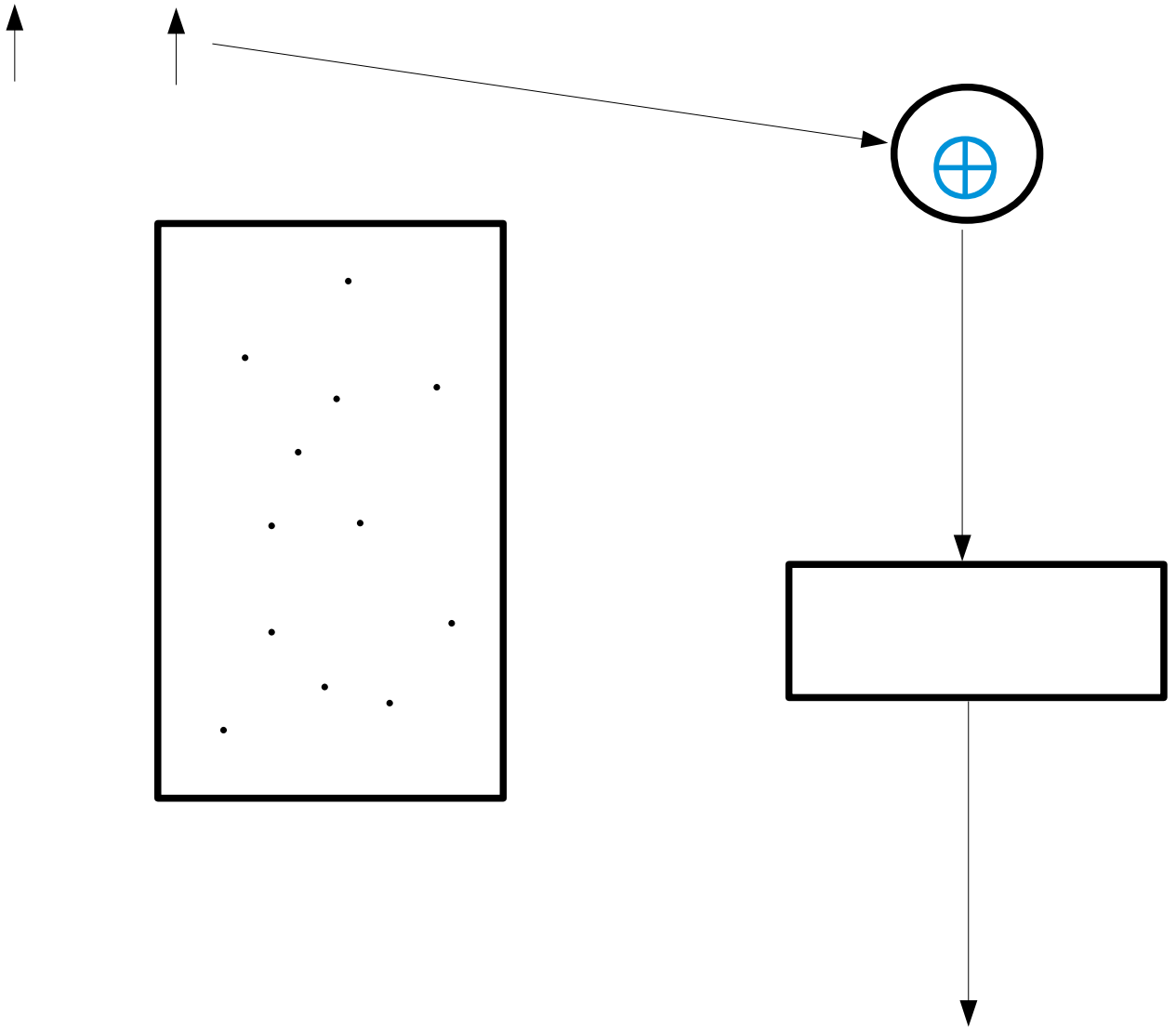
The little brown fox jumped over the lazy dog's back



Secret

Hashing - Encryption

The little brown fox jumped over the lazy dog's back



Secret

To receiver