University of Nijmegen

Hashcash

A Denial of Service counter-measure

Jaap-Henk Hoepman
Department of Computer Science
University of Nijmegen, the Netherlands
jhh@cs.kun.nl
www.cs.kun.nl/~jhh

University of Nijmegen

Reference

Adam Back

Hashcash – A Denial of Service Counter-measure

http://www.cypherspace.org/hashcash/hashcash.pdf

University of Nijmegen

Contents

- Denial of service attacks.
- Cost functions.
- Hashcash.
- Interactive hashcash.
- Applications.
- Cost function classification.
- Conclusions.

University of Nijmegen

DoS attacks

- Spamming.
- Network flooding:
  - *TCP SYN flooding.*
  - *shopping basket depletion.*
- CPU overloading.

University of Nijmegen

Countering DoS attacks

Mounting a DoS attack is ”free”.

$\Downarrow$

We need to incur a cost on a client (for accessing a service).

- Money:
  - *e-cash, micropayment.*
- Resources (CPU time, network bandwidth):
  - *client puzzles / cost functions.*

University of Nijmegen

Cost functions

Principle: let client compute/"mint" a token, and verify result at the server.

Requirements:

- Efficiently verifiable
- (Parameterisably) Expensive to compute

interactive

$$c \leftarrow \text{challenge}(s, w)$$
$$t \leftarrow \text{mint}(c)$$
$$v \leftarrow \text{value}(t)$$

or

non-interactive

$$t \leftarrow \text{mint}(s, w)$$
$$v \leftarrow \text{value}(t)$$

University of Nijmegen

Cost function types

- Publicly auditable:
  - *Efficiently verifiable by any thrid party.*
- Fixed vs. probabilistic cost:
  - *Bounded vs. unbounded.*
- Trapdoor free:
  - *Server cannot cheaply mint tokens.*
- Non-parallelizable.

University of Nijmegen

Hashcash

A non-interactive, publicly auditable, trapdoor-free cost function.

For bitstrings $s \in \{0,1\}^*$, let s_i be the i-th bit of s (s_1 being the leftmost bit). Define

$$x =_b y \quad \triangleq \quad (\forall i : 1 \leq i \leq b :: x_i = y_i)$$

Hashcash function definition (based on a given hash function h with k output bits) for server s with workload w.

- mint(): find $x \in \{0,1\}^*$ such that $h(s\|x) =_w 0^k$. Return $\langle s, x \rangle$.
- value(): find maximal v such that $h(s\|x) =_v 0^k$. Return v.

University of Nijmegen

Hashcash: notes

- For "good" h, computing a token requires brute force search, taking on average 2^{w-1} tries.
- Everybody can verify a token.
- But the server needs to prevent double spending:
  - *Store tokens in database.*
  - *Limit validity of tokens by including timestamp in service name.*
- Non-interactive ⇒ precomputing tokens possible.
  - *Beacons.*
  - *Interactive hashcash.*

University of Nijmegen

Interactive hashcash

Interactive hashcash function definition (based on a given hash function h with k output bits) for server s with workload w.

- challenge(): choose $c \in \{0,1\}^k$. Return $\langle s, w, c \rangle$.
- mint(): find $x \in \{0,1\}^*$ such that $h(s\|c\|x) =_w 0^k$. Return $\langle s, x \rangle$.
- value(): find maximal v such that $h(s\|c\|x) =_v 0^k$. Return v.

University of Nijmegen

Applications

- combatting SPAM
- countering DoS attacks

University of Nijmegen

SPAM

Include a hashcash token in every mail message sent.

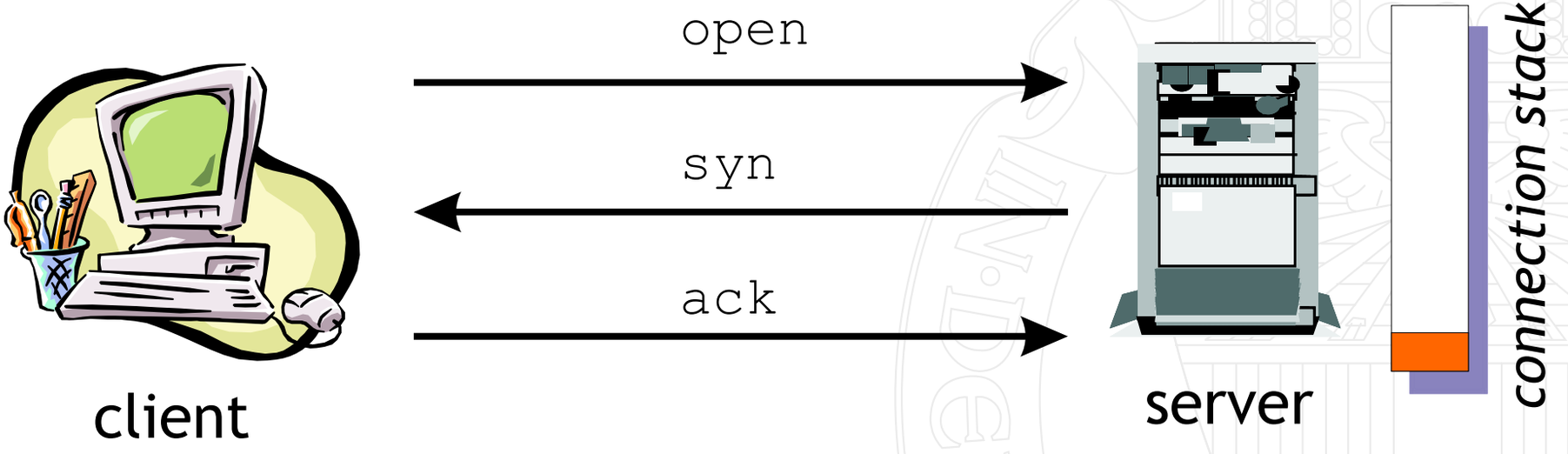
- Non-interactive.
  - *(unless SMTP protocol is changed).*
- s equals mail address.
- Include time in s, or use challenge broadcast by a beacon.
  - *To combat pre-computation attacks.*

University of Nijmegen

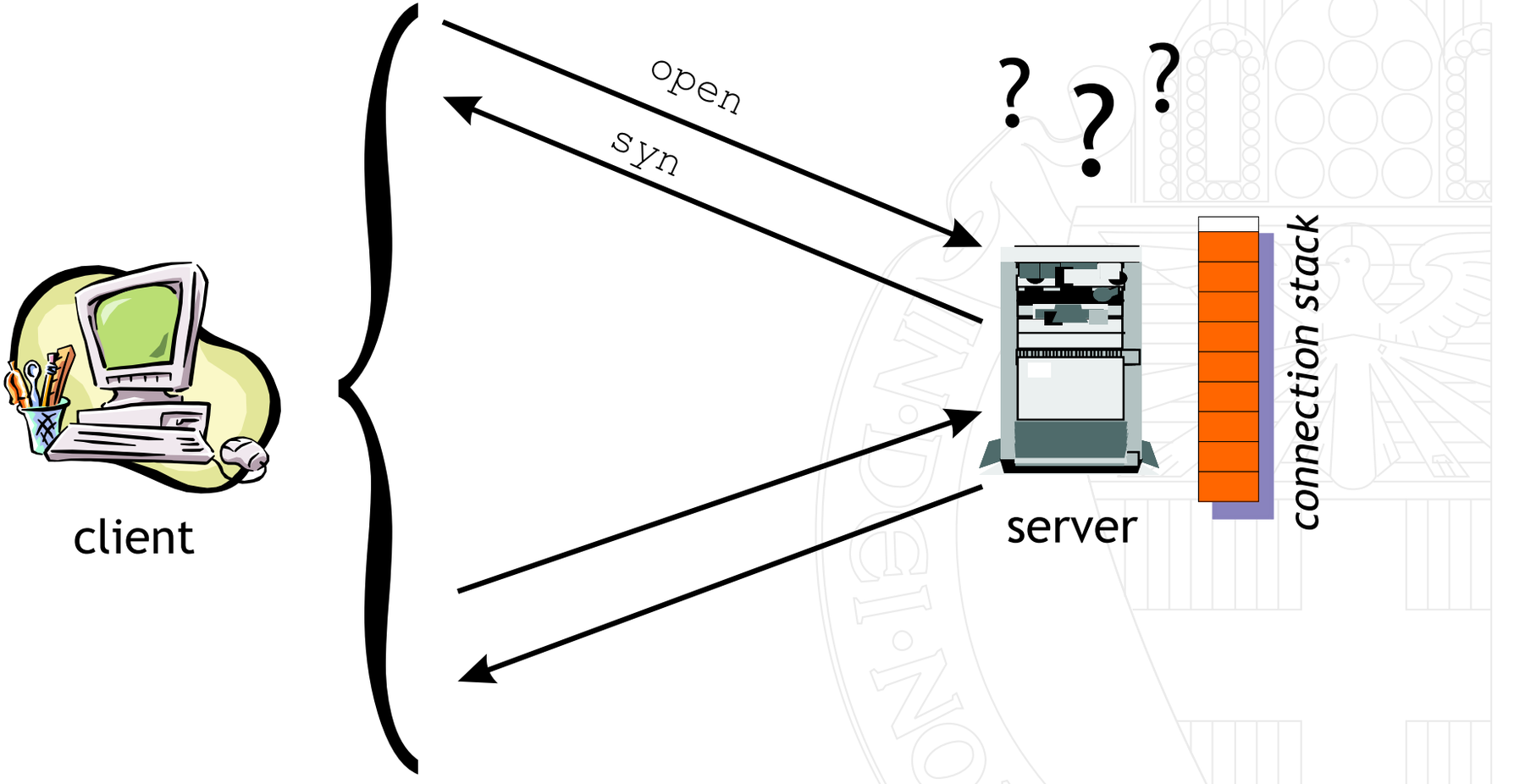
SPAM: discussion

- Determining acceptable workload: spammer with fast hardware vs. ordinary user with slow hardware.
  - *Compute token while composing message.*
  - *Sending to mailinglists (or Cc:-ing) becomes a problem?*
- Standardisation.

DoS: TCP/IP connection setup



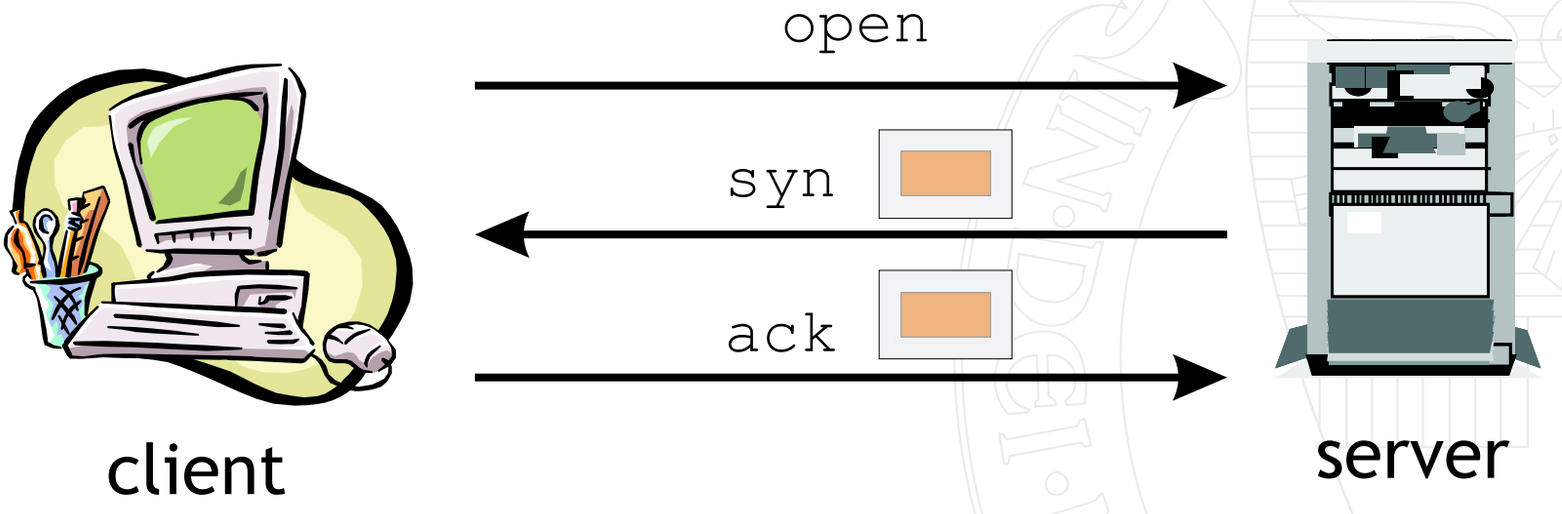
DoS: A SYN flooding attack



DoS: Using SYN-cookies

Problem: the server maintains state for every request.

Solution:



University of Nijmegen

DoS: Using hashcookies

Problem: Fully open TCP connections still consume space, if only at the application layer.

Solution: Require an interactive hashcash token for setting up a connection (to be sent with ACK message). The challenge is sent with the SYN message.

Note: Use approach similar to SYN-cookies, to avoid storing state for half-open connections.

University of Nijmegen

DoS: Dynamic throttling

- Increase workfactor with increasing load on server.
- Backwards compatibility: require hashcookies after reaching certain load.

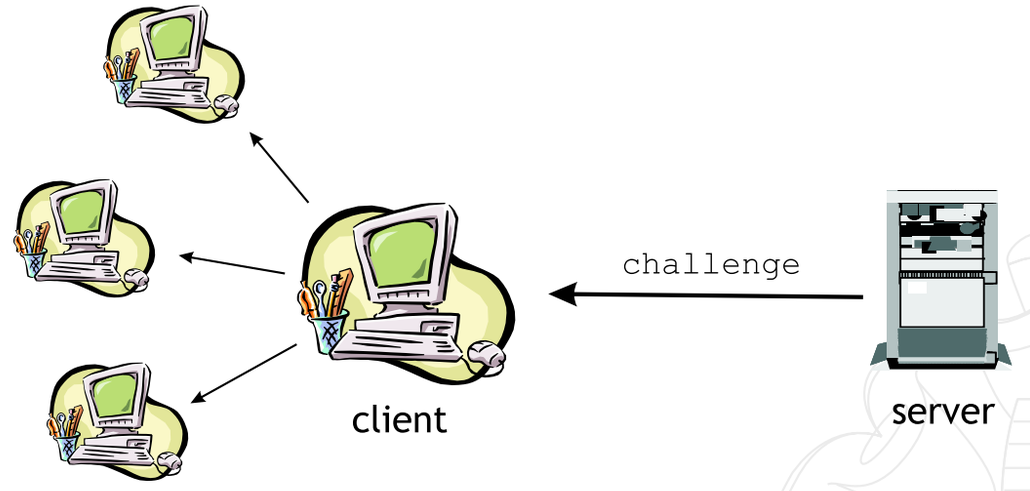
University of Nijmegen

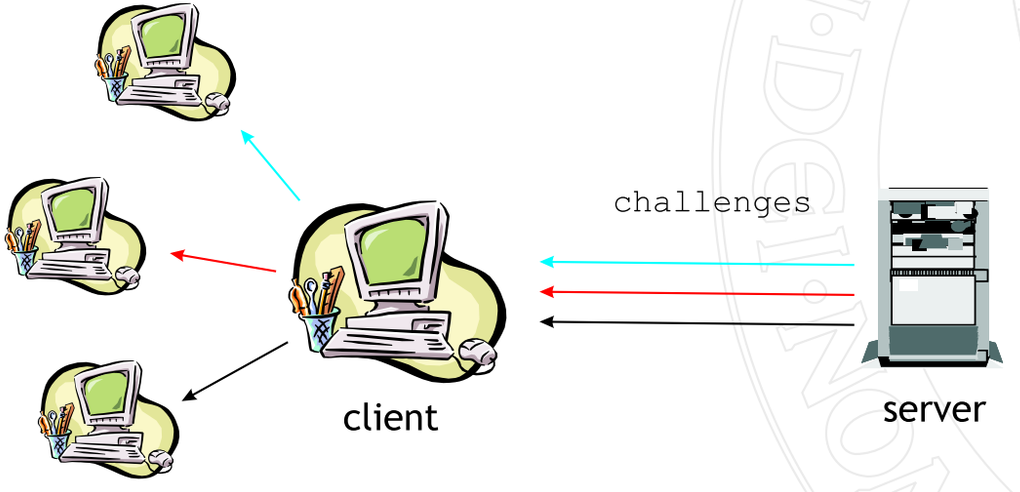
A non-parallelizable cost function

Based on Rivest *et. al.* time-lock puzzles. Let $n = pq$ be public. Let $\varphi(n) = (p-1)(q-1)$ be known to the server only.

- challenge(): choose $c \in \{0,1\}^k$. Return $\langle s, w, c \rangle$.
- mint(): compute $x \leftarrow h(s,c)$, then compute $y \leftarrow x^{x^w} \bmod n$ and return $\langle s, c, w, y \rangle$
  - *$\varphi(n)$ is unknown: requires w exponentiations.*
- value(): compute $x \leftarrow h(s,c)$, then compute $z \leftarrow x^w \bmod \varphi(n)$. Return w if $x^z = y \bmod n$, otherwise return 0.
  - *$\varphi(n)$ is known: requires 2 exponentiations.*

Parallelisability: a non-issue!





University of Nijmegen

Open problems

The following types of cost-functions are unknown to exist:

- Efficiently-verifiable non-interactive fixed-cost.
  - *(hashcash has a probabilistic cost).*
- Efficiently-verifiable non-interactive non-parallelizable.
  - *(time-lock puzzles are just* practically *verifiable).*
- Publicly-auditable non-interactive fixed-cost.
  - *(hashcash has a probabilistic cost).*

(where efficient means: equivalent to the cost of applying a hash function h).

University of Nijmegen

Conclusions

- Cost-functions can be used to mitigate DoS attacks.
- Practical applicability remains to be determined.