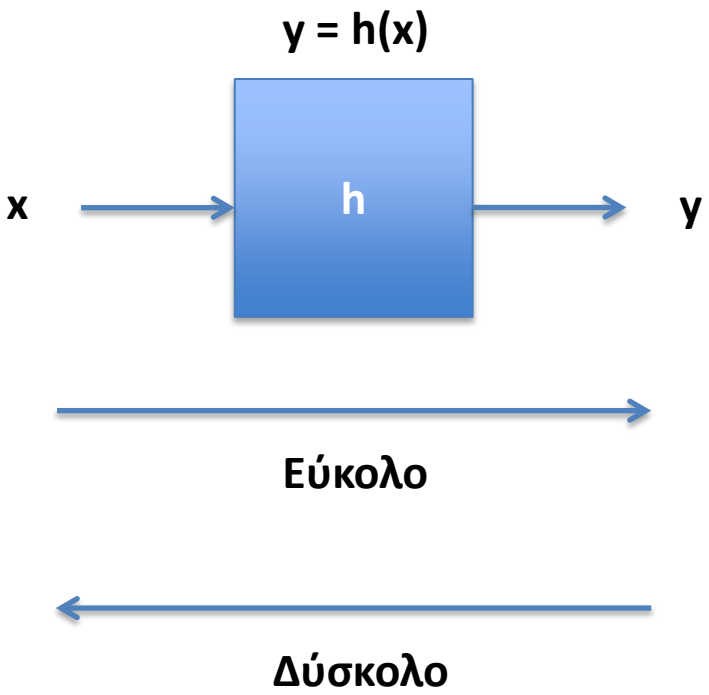
Κρυπτογραφία: Εφαρμογές Hash

Διαφάνειες: Δ. Ζήνδρος

Στόχοι του σημερινού μαθήματος

- Επεκτάσεις του collision resistance
- Δεσμεύσεις / hiding / binding
- Αλάτι
- md5, sha1, sha256, bcrypt, scrypt
- Proof of work
- Δέντρα Merkle
- Αποθήκευση κωδικών πρόσβασης
- Timestamping

One-way functions



Hash functions

- Στην κρυπτογραφία, τα “hash functions” πρέπει να έχουν δυσκολία αντιστροφής
- h(x) = y μπορεί να υπολογιστεί πολυωνυμικά
- Το αντίστροφο όμως όχι

Collision resistance

- **Preimage**: Δεδομένου y, είναι **δύσκολο** να βρεθεί x που y = h(x)
- **2nd preimage**: Δεδομένου x, είναι **δύσκολο** να βρεθεί x' που h(x) = h(x')
  - Είναι δύσκολο ν'αλλάξει το x χωρίς ν'αλλάξει το y
- **Collision resistance**: Είναι **δύσκολο** να βρεθούν x_1 και x_2 τέτοια ώστε $h(x_1) = h(x_2)$

Collision resistance

- **Perturbation**: Είναι **δύσκολο** να βρεθούν x, x' τέτοια ώστε:
  - $y = H(x)$
  - $y' = H(x')$
  - $\text{Hamming}(y, y') < \varepsilon$
- **Range**: Δεδομένου y και ε, είναι δύσκολο να βρεθεί x τέτοιο ώστε $y \leq H(x) < y + \varepsilon$. Για $y = 0$, αναζητείται x τέτοιο ώστε $H(x) < \varepsilon$.

md5

- Δημοφιλής μέθοδος hashing
- $\{0, 1\}^* \rightarrow \{0, 1\}^{128}$
- **Δεν** είναι κρυπτογραφικά ασφαλές
- Γνωρίζουμε collisions
- **Μην** το χρησιμοποιείτε για κρυπτογραφικές εφαρμογές

Παράδειγμα md5

md5('Hello world')

=

3e25960a79dbc69b674cd4ec67a72c62

md5 collision

md5(d131dd02c5e6eec4693d9a0698aff95c 2fcab58712467eab4004583eb8fb7f89
55ad340609f4b30283e488832571415a 085125e8f7cdc99fd91dbdf280373c5b
d8823e3156348f5bae6dacd436c919c6 dd53e2b487da03fd02396306d248cda0
e99f33420f577ee8ce54b67080a80d1e c69821bcb6a8839396f9652b6ff72a70)

=

md5(d131dd02c5e6eec4693d9a0698aff95c 2fcab50712467eab4004583eb8fb7f89
55ad340609f4b30283e4888325f1415a 085125e8f7cdc99fd91dbd7280373c5b
d8823e3156348f5bae6dacd436c919c6 dd53e23487da03fd02396306d248cda0
e99f33420f577ee8ce54b6708280d1e c69821bcb6a8839396f965ab6ff72a70)

sha1

- Δημοφιλής μέθοδος hashing
- $\{0, 1\}^* \rightarrow \{0, 1\}^{160}$
- **Δεν** είναι κρυπτογραφικά ασφαλές
- Δεν γνωρίζουμε collisions
  - Όμως έχουμε “free start collisions” (8 Οκτωβρίου 2015)
  - Γνωρίζουμε επίθεση πολυπλοκότητας 2^{61}
- **Μην** το χρησιμοποιείτε για κρυπτογραφικές εφαρμογές

Παράδειγμα sha1

sha1('Hello world')

=

7b502c3a1f48c8609ae212cdfb639dee39673f5e

sha2

- Μετεξέλιξη του sha1
- Θεωρείται **ασφαλές** για κρυπτογραφικές εφαρμογές
- SHA224: $\{0, 1\}^* \rightarrow \{0, 1\}^{224}$
- **SHA256: $\{0, 1\}^* \rightarrow \{0, 1\}^{256}$**
- SHA384: $\{0, 1\}^* \rightarrow \{0, 1\}^{384}$
- SHA512: $\{0, 1\}^* \rightarrow \{0, 1\}^{512}$

Παράδειγμα sha256

sha256('Hello world')

=

64ec88ca00b268e5ba1a35678a1b5316d212f4f3
66b2477232534a8aeca37f3c

Χρησιμοποιώντας τη βιβλιοθήκη OpenSSL

```
dionyziz@erdos ~ % echo -n 'Hello world'|openssl md5
(stdin)= 3e25960a79dbc69b674cd4ec67a72c62
dionyziz@erdos ~ % echo -n 'Hello world'|openssl sha1
(stdin)= 7b502c3a1f48c8609ae212cdfb639dee39673f5e
dionyziz@erdos ~ % echo -n 'Hello world'|openssl sha256
(stdin)= 64ec88ca00b268e5ba1a35678a1b5316d212f4f366b2477232534a8aeca37f3c
dionyziz@erdos ~ %
```

«Αντιστρέφοντας» ένα hash

- Τα hashes είναι σχεδιασμένα να μην είναι αντιστρέψιμα
- Συνεπώς η αντιστροφή μπορεί να γίνει **μόνο** χρησιμοποιώντας **brute force**
  - Δοκιμάζουμε όλα τα πιθανά strings μήκους 0, 1, 2, 3, κλπ. από ένα δεδομένο αλφάβητο
  - Δοκιμάζουμε λέξεις από ένα **λεξικό**

Dictionary attack σε hash

```
foreach (word in dictionary) {
    if (SHA256(word) == c) {
        return word;
    }
}
```

Brute-force attack σε hash

```
Σ = {'a', 'b', …, 'z'};
numdigits = 0;
m = "";
while (SHA256(m) != c) {
    try {
        m = increment(m, Σ, numdigits);
    }
    catch (OutOfBoundsException e) {
        ++numdigits;
        m = repeat(Σ[0], numdigits);
    }
}
return m;
```

Σχήματα δέσμευσης

- Η Alice θέλει να δεσμευτεί σε κάποια τιμή b
- Η Alice και ο Bob δεν εμπιστεύονται ο ένας τον άλλον
- **Binding**: Ο Bob θέλει να γνωρίζει ότι η Alice δεν θα αλλάξει τη γνώμη της για το b
- **Hiding**: Η Alice δεν θέλει ακόμα να αποκαλύψει την τιμή της τώρα, αλλά αργότερα

Σχήματα δέσμευσης

Δύο στάδια:

- Φάση δέσμευσης:
  - Η Alice διαλέγει και **δεσμεύεται** σε μία τιμή
  - Στέλνει ένα μυστικό στο Bob
  - **Hiding**: Ο Bob δεν μπορεί να χρησιμοποιήσει το μυστικό για να βρει την τιμή στην οποία έχει δεσμευθεί η Alice
- Φάση αποκάλυψης:
  - Ο Bob **μαθαίνει** την αρχική τιμή στην οποία είχε δεσμευθεί η Alice
  - **Binding**: Ο Bob επιβεβαιώνει ότι η Alice δεν άλλαξε την τιμή της

Απλή δέσμευση με hashes

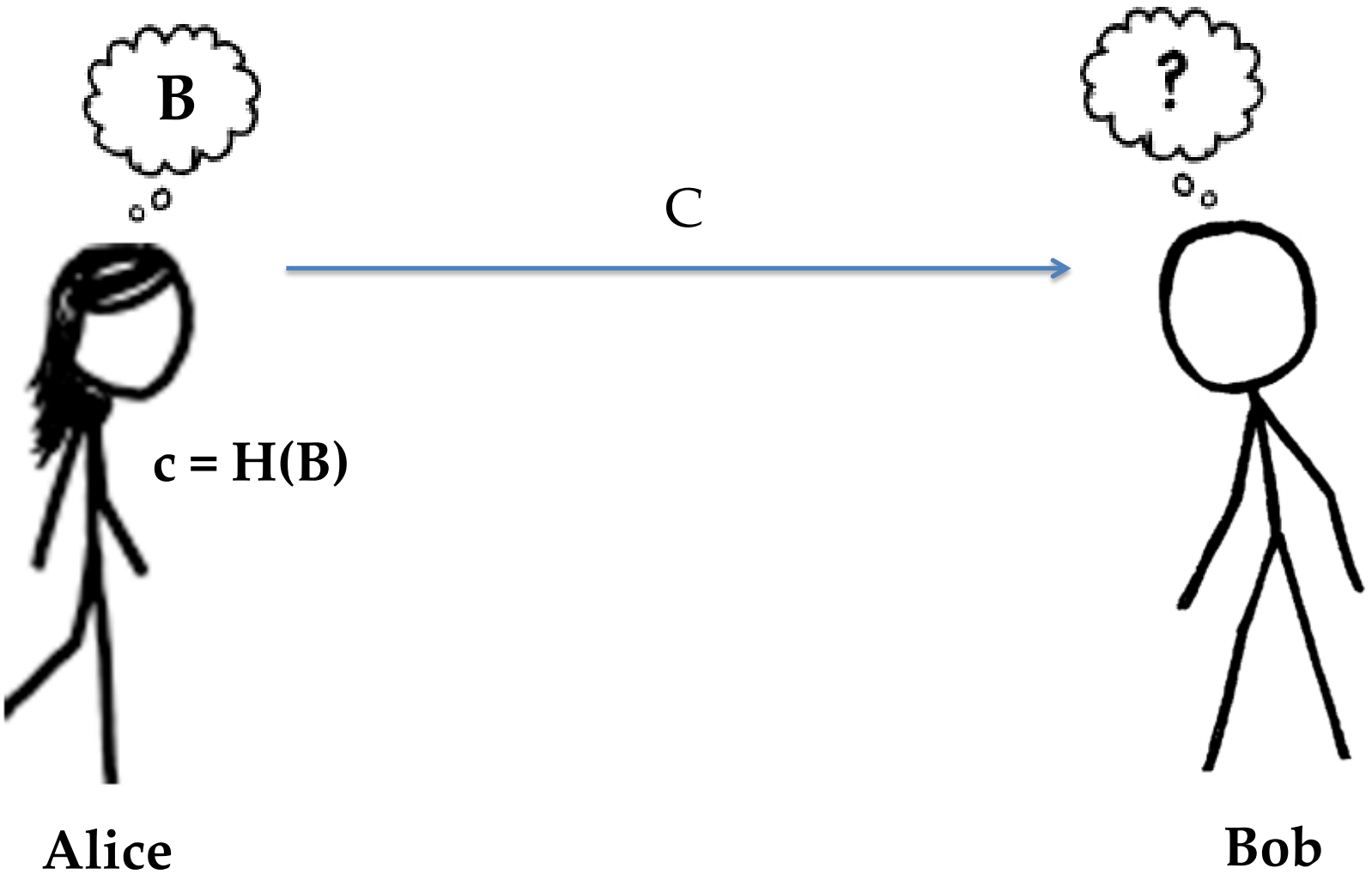
Φάση δέσμευσης:

- Η Alice υπολογίζει το c = H(b)
- Στέλνει το c στον Bob.
- Ο Bob βλέπει το c. Λόγω του ότι η Η είναι κρυπτογραφικά ασφαλής hash function, ο Bob δεν μπορεί να επιστρέψει στο b.

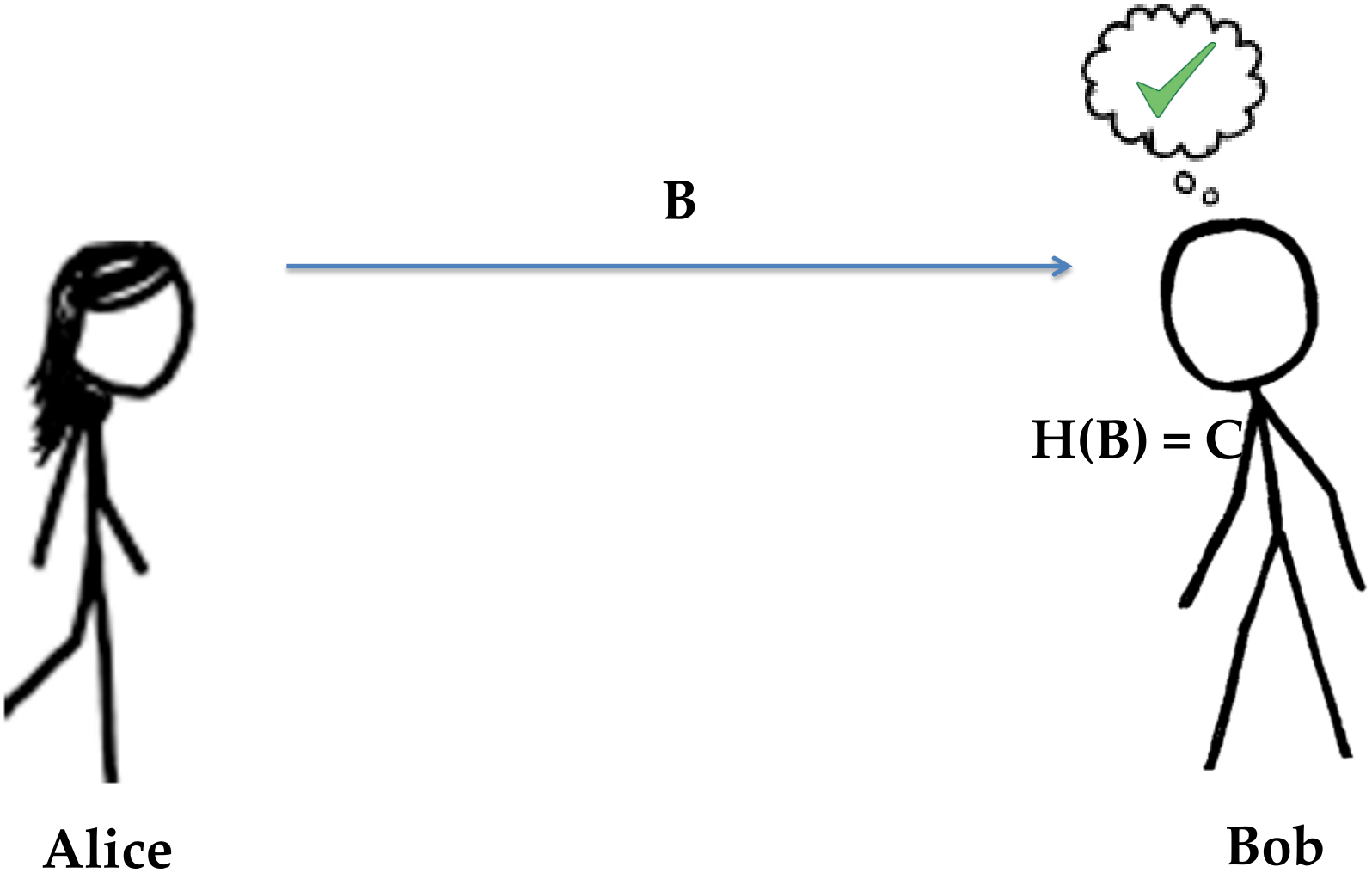
Φάση αποκάλυψης:

- Η Alice στέλνει το b στον Bob
- Ο Bob ελέγχει ότι H(b) = c.

Φάση δέσμευσης



Φάση αποκάλυψης



Είναι πραγματικά hiding;

Τι γίνεται αν ο Bob μπορεί να μαντέψει το b?

- Έστω ότι οι πιθανές τιμές του b είναι 0 και 1
- Ο Bob μπορεί να δοκιμάσει αυτές τις τιμές:

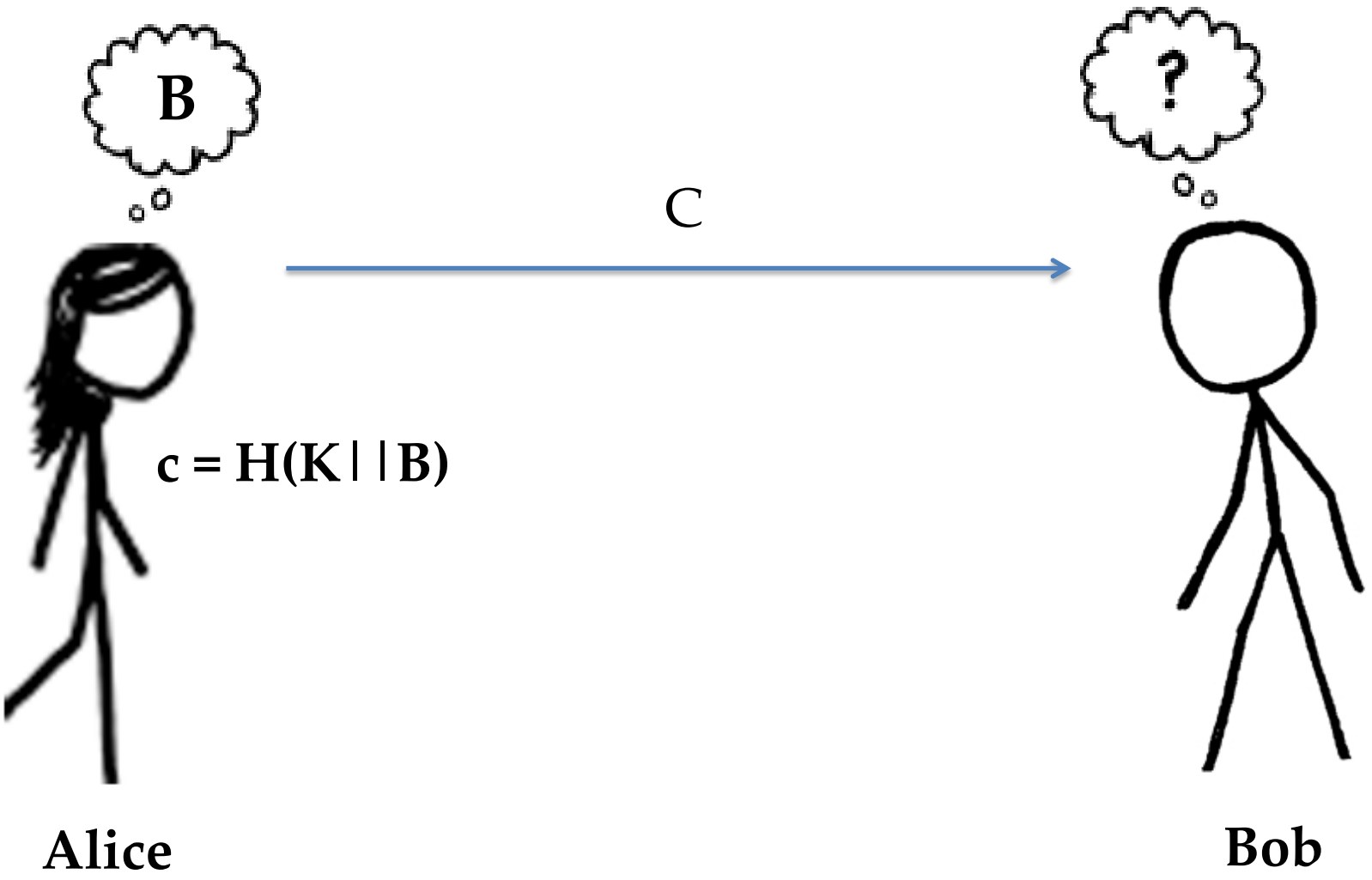
  H(0) = c?

  H(1) = c?
- Έτσι μπορεί να αποκαλύψει το b
- Πώς μπορούμε να αμυνθούμε σε αυτό;

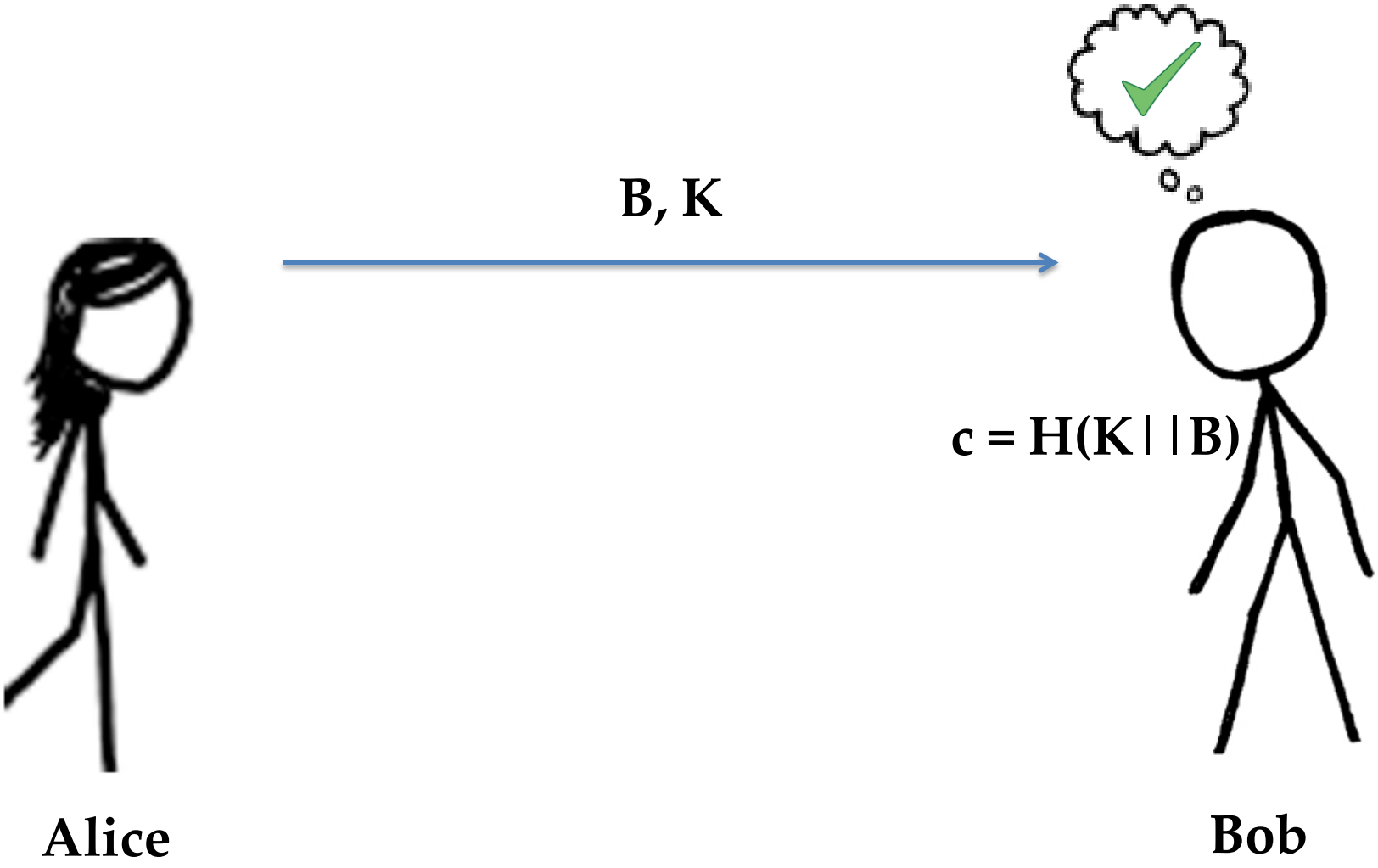
Αλάτι (salt)

- Η Alice διαλέγει ένα μυστικό k
- Υπολογίζει το c = H(k||b)
- Στέλνει στον Bob το c
- Στην αποκάλυψη, στέλνει το k και το b
- Το k ονομάζεται **αλάτι**

Φάση δέσμευσης (commit)



Φάση αποκάλυψης (reveal)



Ρίψη νομίσματος

- Η Alice και ο Bob θέλουν να παίξουν κορόνα-γράμματα
- Όμως είναι μακριά
- Δεν εμπιστεύονται ο ένας τον άλλον
- Πώς θα μπορούσαν να παίξουν;
  - Δε θα μπορούσε απλά η Alice να εμπιστευθεί τον Bob να ρίξει ένα φυσικό νόμισμα και να της πει τι έφερε!

Ρίψη νομίσματος

- Ρίψη νομίσματος: Θέλουμε να δημιουργήσουμε πιθανότητα 50% κέρδους για την Alice και 50% για τον Bob
- Η Alice διαλέγει έναν αριθμό A στο {0, 1}
- Ο Bob διαλέγει έναν αριθμό B στο {0, 1}
- Αν $A \oplus B = 0$, τότε κερδίζει η Alice
  - 00, 11: Πιθανότητα 50%
- Αν $A \oplus B = 1$, τότε κερδίζει ο Bob
  - 01, 10: Πιθανότητα 50%

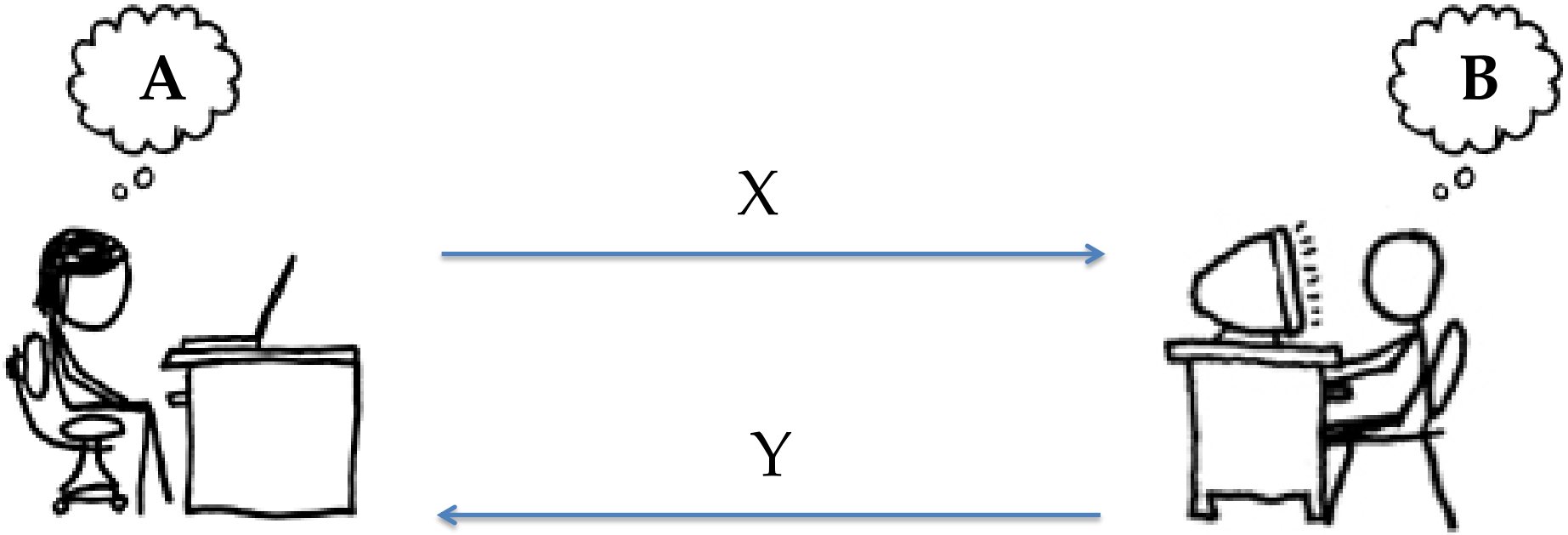
Ρίψη νομίσματος

- Όμως ποιο θα αποκαλυφθεί πρώτα; Το Α ή το Β;
- Αν η Alice αποκαλύψει πρώτη το Α, τότε ο Bob μπορεί να **αλλάξει** την επιλογή του Β ώστε να κερδίσει

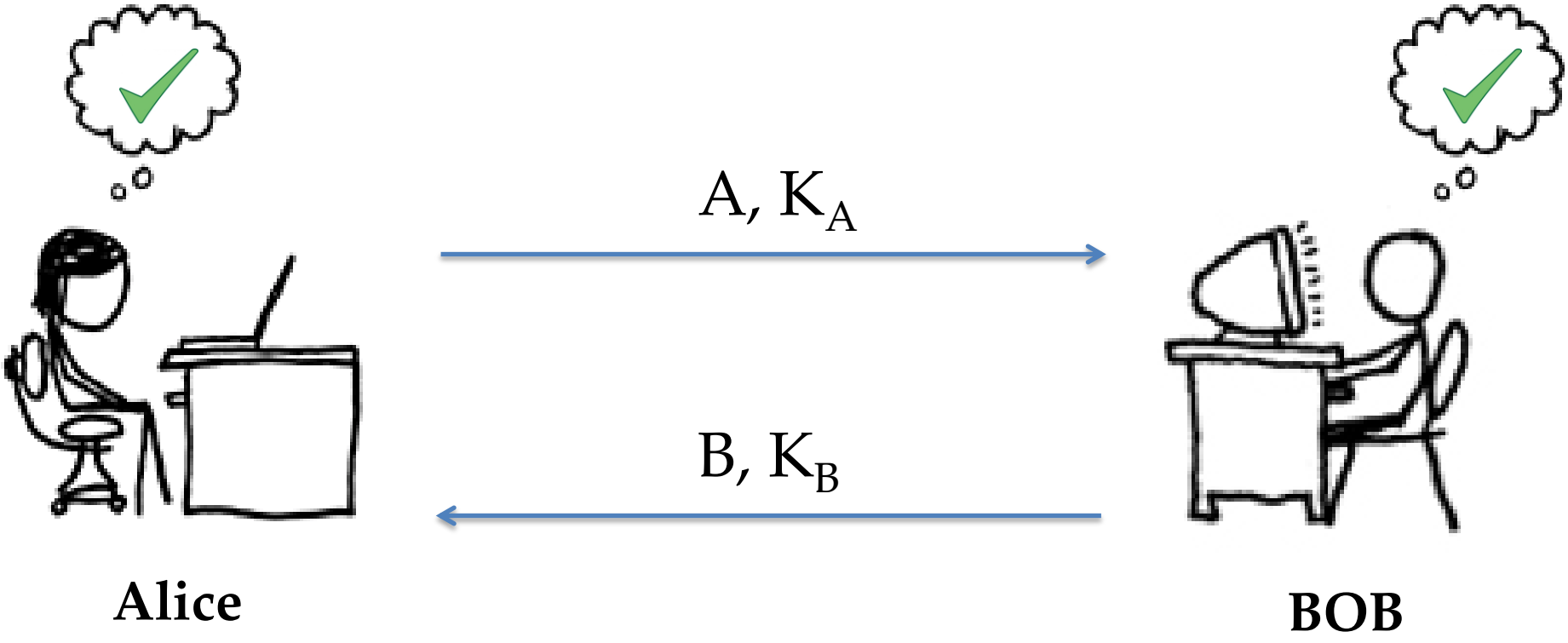
Ορθή ρίψη νομίσματος

- Η Alice διαλέγει έναν αριθμό a στο {0, 1}
- Η Alice δεσμεύεται στο A με δέσμευση X και αλάτι K_A
  - $X = SHA256(A||K_A)$
- Ο Bob διαλέγει έναν αριθμό b στο {0, 1}
- Ο Bob δεσμεύεται στο B με δέσμευση Y και αλάτι KB
  - $Y = SHA256(B||K_B)$
- Ανταλλάσσουν τα X και Y με οποιαδήποτε σειρά
- Αποκαλύπτουν τα A και B με οποιαδήποτε σειρά

Ρίψη νομίσματος: Δέσμευση



Ρίψη νομίσματος: Αποκάλυψη



Ρίψη νομίσματος

Winner = A ⊕ B

0 = Alice

1 = BOB

Ρήψη νομίσματος

- Υπάρχει πρόβλημα αν η Alice και ο Bob θέλουν να στοιχηματίσουν χρήματα στο ποιος θα κερδίσει στη ρίψη;

Ρήψη νομίσματος

- Υπάρχει πρόβλημα αν η Alice και ο Bob θέλουν να στοιχηματίσουν χρήματα στο ποιος θα κερδίσει στη ρίψη;
- Ναι - Όποιος χάσει μπορεί να **αποχωρήσει** χωρίς να πληρώσει
  - Θα λύσουμε αυτό το πρόβλημα κρυπτογραφικά σε επόμενο μάθημα, αφότου μιλήσουμε για bitcoin

Timestamping

- Έστω ότι θέλουμε να εγγυηθούμε ότι...
  - Μία εφεύρεση δημιουργήθηκε **πριν** από κάποια συγκεκριμένη ημερομηνία
  - Και από κάποιον συγκεκριμένο εφευρέτη
  - Χωρίς όμως να αποκαλυφθεί η εφεύρεση
  - Η εφεύρεση περιγράφεται σε ένα κείμενο
- Πώς θα μπορούσε να γίνει κάτι τέτοιο;

Απόδειξη copyright

- Παραδοσιακή μέθοδος: Δημοσιεύω ένα αναγραμματισμό της εφεύρεσης
  - Νόμος του Hooke, 1660: "ceiiinosssttuv"
  - ut tensio sic vis (όσο η προέκταση, τόση και η δύναμη)
- Σύγχρονη μέθοδος: **Δημοσιεύω** ένα hash του κειμένου που περιγράφει την εφεύρεση και τον δημιουργό της.
- Για επιβεβαίωση, δημοσιεύω το αρχικό κείμενο
- Η αρχαιότερη δημοσίευση hash νικά
- Ιδέα ιδέα με το commit / reveal

Δημοσιεύοντας

- Πού μπορεί να γίνει μία τέτοια δημοσίευση;
  - Σε κάποια έμπιστη εφημερίδα π.χ. NY Times
  - Πιο εύκολα, στο bitcoin blockchain
  - Θα μελετήσουμε απόδειξη copyright μέσω bitcoin αργότερα μέσα στο εξάμηνο

Timestamping

- Έστω ότι θέλουμε να εγγυηθούμε ότι...
  - Ένα συγκεκριμένο hash δημιουργήθηκε **μετά** από κάποια ημερομηνία
- Πώς θα μπορούσε να γίνει κάτι τέτοιο;

Timestamping

- Στο preimage του hash περιλαμβάνω κάποια πληροφορία που δεν ήταν γνωστή μέχρι εκείνη την ημερομηνία

π.χ.

```
H("The Times 03/Jan/2009 Chancellor on brink of second bailout for banks"||data)
```


THE TIMES

£1.50

Eat Out from £5

More than 900 great restaurants including four Gordon Ramsay favourites from £15

Israel prepares to send tanks and troops into Gaza

Chancellor on brink of second bailout for banks

99p

Michael Sheen
Frost, Nixon and me

Working mums
So that's how she does it

Detox in style
The best spas on the planet

Salman Rushdie
I won't marry again

Giant killing?
Guide to the FA Cup third round

Άσκηση

- Να αντιστρέψετε ένα SHA256
  - Δεδομένου y, βρείτε ένα x τέτοιο ώστε H(x) = y
  - Μα αυτό είναι αδύνατο?!?


a grand don't come for free
THE STREETS
SHA256 SHA256 SHA256 SHA256 SHA256
SHA256 SHA256 SHA256 SHA256 SHA256
SHA256 SHA256 SHA256 SHA256 SHA256

Διάλλειμα

Ασφαλής αποθήκευση κωδικών

- Έχουμε μία εφαρμογή με χρήστες, π.χ. web εφαρμογή ή online video game
- Θέλουμε να αποθηκεύσουμε κωδικούς πρόσβασης χρηστών
- Μπορούμε να τους αποθηκεύσουμε σαν plaintext σε μία βάση δεδομένων
- Τι πρόβλημα έχει αυτό;

Ασφαλής αποθήκευση κωδικών

- Αποθήκευση σε plaintext: Αν κλαπεί η βάση (λόγω παραβίασης των συστημάτων), τότε ο θύτης αποκτά πρόσβαση **σε όλους τους κωδικούς**
- Μπορούμε να τους **κρυπτογραφήσουμε**;

Ασφαλής αποθήκευση κωδικών

- Αποθήκευση κρυπτογραφημένα: Πρέπει να έχουμε το **κλειδί** για να τους **αποκρυπτογραφήσουμε** ώστε να επιτρέψουμε logins
- Το **κλειδί** μπορεί να υποκλαπεί επίσης!
- Συνεπώς είναι κακή ιδέα

Ασφαλής αποθήκευση κωδικών

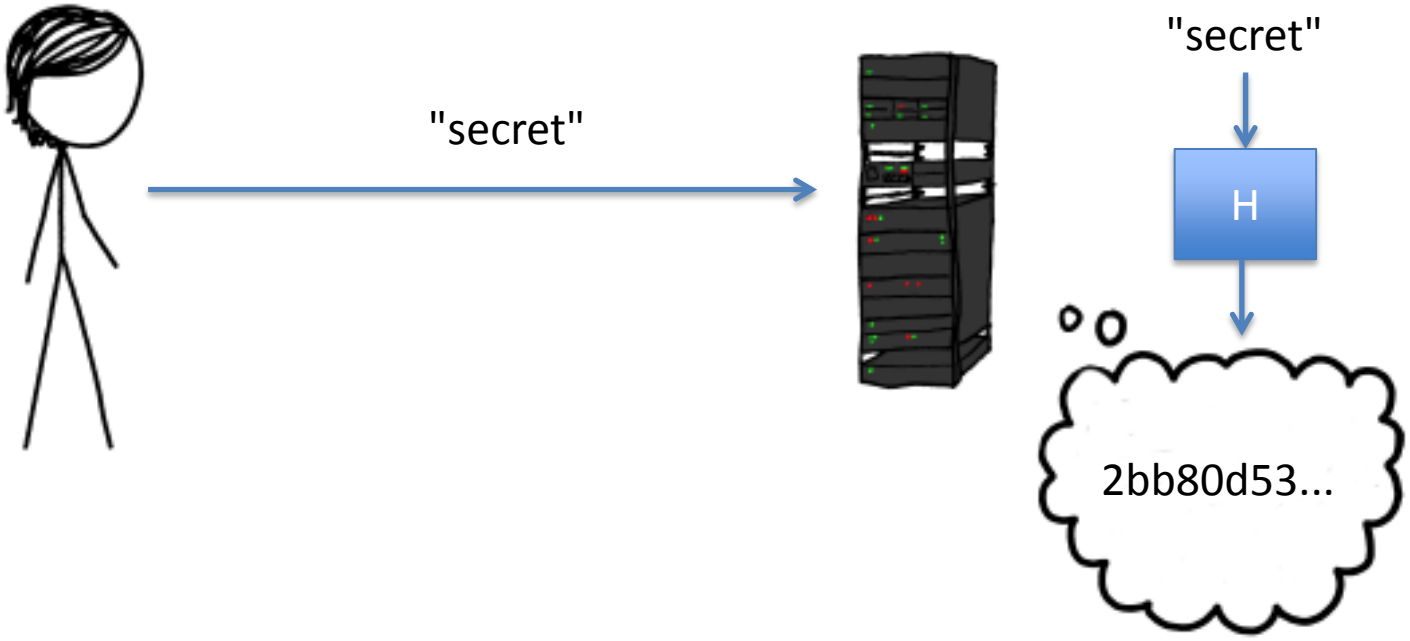
Δημιουργία λογαριασμού:

- Ο χρήστης στέλνει το password
- Αποθηκεύουμε το c = H(password)

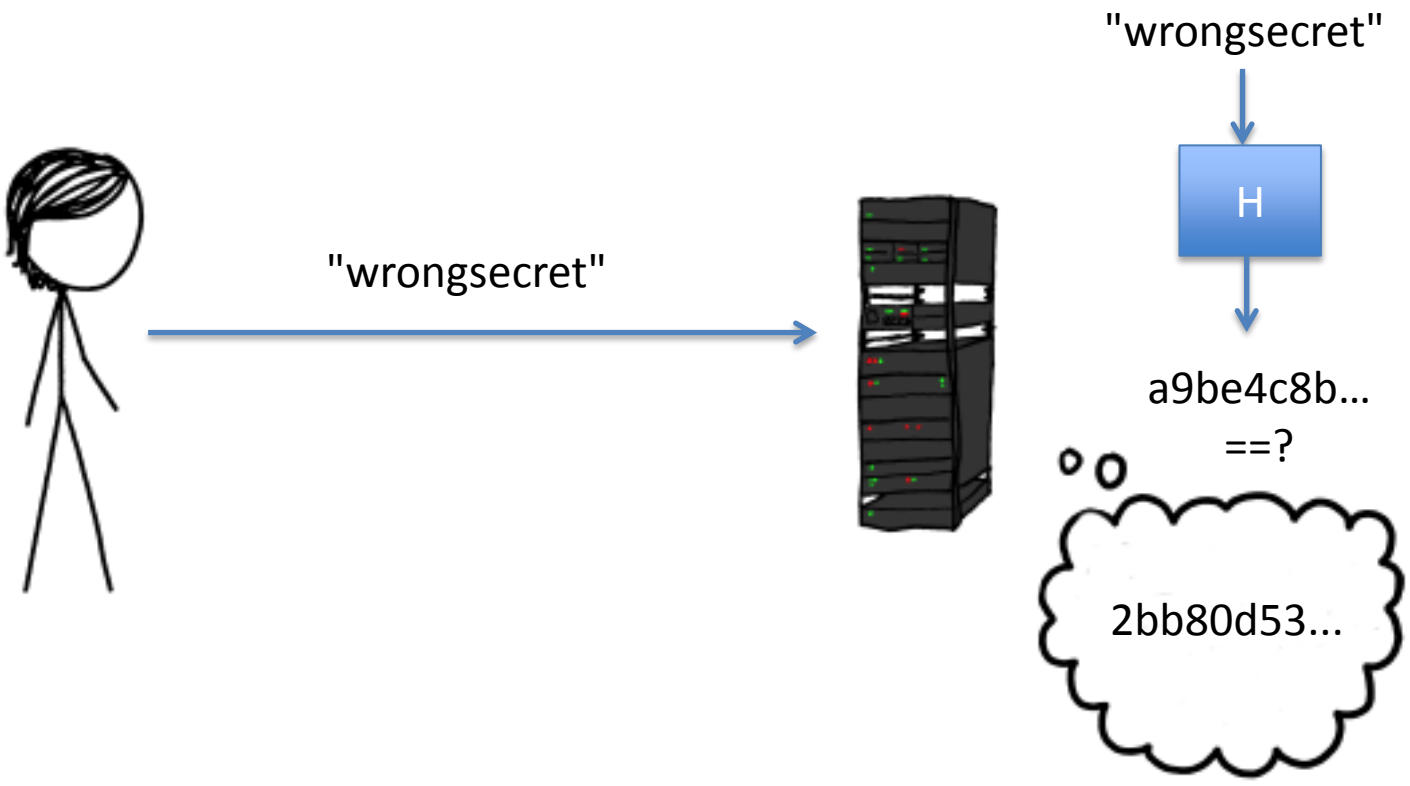
Login:

- Ο χρήστης στέλνει το password
- Υπολογίζουμε το H(password) ξανά
- Το συγκρίνουμε με το c αποθηκευμένο στη βάση

Δημιουργία λογαριασμού



Login



Ασφαλής αποθήκευση κωδικών

- Θα ήταν σωστό να ζητούσαμε από το χρήστη να κάνει hash και να μας στείλει τον κωδικό του στο login;

Ασφαλής αποθήκευση κωδικών

- Θα ήταν σωστό να ζητούσαμε από το χρήστη να κάνει hash και να μας στείλει τον κωδικό του στο login;

- Όχι - τότε η υποκλοπή του hash είναι ισοδύναμη με κωδικό, αποκτά κανείς πρόσβαση

	UserID	UserName	Password
1	1	Scott	0x3858F62230AC3C915F300C664312C63F
2	2	John	0xAE2D699ACA20886F6BED96A0425C6168
3	4	Frank	0x059BF68F71C80FCE55214B411DD2280C
4	5	Ezra	0xFF8B689ED9E3D6E3A22BB0357384C770
5	6	Jisun	0x2B898208A66F66447F49FD0532C01AEB

Σπάσιμο πολλών hashes

- Έστω ότι θέλουμε να αντιστρέψουμε πολλά hashes που έχουμε υποκλέψει
- Τι θα μπορούσαμε να κάνουμε;

Σπάσιμο πολλών hashes

- Έστω ότι θέλουμε να αντιστρέψουμε πολλά hashes που έχουμε υποκλέψει
- Τι θα μπορούσαμε να κάνουμε;
- **Lookup table**
  - Μόνιμη αποθήκευση όλων των H(x) για τα υποψήφια x (π.χ. έως 6 χαρακτήρες)
  - Κάθε φορά που θέλουμε να σπάσουμε ένα hash, κοιτάμε αν υπάρχει αποθηκευμένο

Σπάσιμο πολλών hashes

- **Rainbow table**
  - Μόνιμη αποθήκευση όλων των H(x) για τα υποψήφια x (π.χ. έως 6 χαρακτήρες)
  - "Συμπιεσμένη" μορφή αποθήκευσης
    - Αποθηκεύει έναν αντιπρόσωπο για κάθε σειρά από hashes
  - Καλή ισορροπία ανάμεσα σε χώρο και χρόνο

Ασφαλής αποθήκευση κωδικών

- Το απλό hashing **δεν είναι αρκετό!**
- Διαρρέουν πληροφορίες όπως:
  - Έχει η Alice τον ίδιο κωδικό με τον Bob?
  - Ποιος χρήστης έχει τον πιο συνηθισμένο κωδικό;
- Η δημιουργία ενός {lookup,rainbow} table επιτρέπει σε έναν επιτιθέμενο να επιτεθεί σε πολλές υπηρεσίες ταυτόχρονα

Αλάτι στην αποθήκευση κωδικών

- Προσθέτουμε μία τυχαία τιμή Κ στην αρχή του κωδικού την οποία ονομάζουμε **αλάτι**
- Αποφεύγουμε τα lookup tables και τα rainbow tables
- Χρήσιμο σε περίπτωση που υποκλαπεί η βάση δεδομένων μας και κάποιος θέλει να σπάσει όλους τους κωδικούς

Αλάτι στην αποθήκευση κωδικών

- Είναι αρκετό να έχουμε αλάτι κοινό για όλους τους χρήστες μας;

π.χ. θα μπορούσε το gmail να χρησιμοποιεί κάτι τέτοιο;

c = H("gmail.comGOOGLEsecret" || password)

Αλάτι στην αποθήκευση κωδικών

- Είναι αρκετό να έχουμε αλάτι κοινό για όλους τους χρήστες μας;
- Το κοινό αλάτι μας προστατεύει από επιθέσεις κοινές με άλλες υπηρεσίες
  - Δεν μπορεί κανείς εύκολα να αποφανθεί αν ο χρήστης έχει τον ίδιο κωδικό στο Facebook και στο Gmail αν το Facebook και το Gmail χρησιμοποιούν από ένα διαφορετικό αλάτι
- Όμως αυτό δεν είναι αρκετό! (Γιατί;)

Σωστή αποθήκευση κωδικών

Δημιουργία λογαριασμού

- Δημιουργούμε αλάτι K **διαφορετικό για κάθε χρήστη**
- Αποθηκεύουμε το K σε plaintext
- Αποθηκεύουμε το H(K||password)

Log in

- Υπολογίζουμε το H(K||password) και το συγκρίνουμε με το αποθηκευμένο

Σωστή αποθήκευση κωδικών

- Έτσι αναγκάζουμε τον επιτιθέμενο να κάνει brute force **για κάθε** κωδικό στη βάση δεδομένων μας
- Το αλάτι πρέπει να είναι μεγάλο
  - Διαφορετικά ο επιτιθέμενος μπορεί να δημιουργήσει {lookup,rainbow} tables για κάθε πιθανό αλάτι

bcrypt

- Η bcrypt είναι μία μέθοδος για αποθήκευση κωδικών πρόσβασης
- Προσφέρει δύο συναρτήσεις:
  1. password_hash(password, difficulty)
  2. password_verify(hash, password)
- Προσθέτει αυτόματα αλάτι
- Έχει **παραμετροποιήσιμη** δυσκολία (χρόνος που χρειάζεται για να εκτελεστεί)
  - Ισορροπία ανάμεσα στη δυσκολία του αντιπάλου και στη δική μας ευκολία
- **Προτεινόμενη μέθοδος**

scrypt

- Παρόμοια λογική με την bcrypt
- Επιπλέον, **αποτρέπει** τη δημιουργία ειδικού hardware που μπορεί να "σπάσει" κωδικούς με μαζική παραλληλοποίηση
- Απαιτεί μεγάλες ποσότητες **μνήμης** για να τρέξει
- Συνεπώς δεν παραλληλοποιείται εύκολα σε hardware
- **Προτεινόμενη μέθοδος**

Proof of work

- Κρυπτογραφικό primitive που μας επιτρέπει να αποδείξουμε ότι ξοδέψαμε κύκλους CPU
- Η Alice θέλει να βεβαιωθεί ότι ο Bob θα αφιερώσει ορισμένους κύκλους υπολογιστικής δύναμης μόνο για εκείνη ♥
- Πώς θα μπορούσε να το κάνει αυτό;

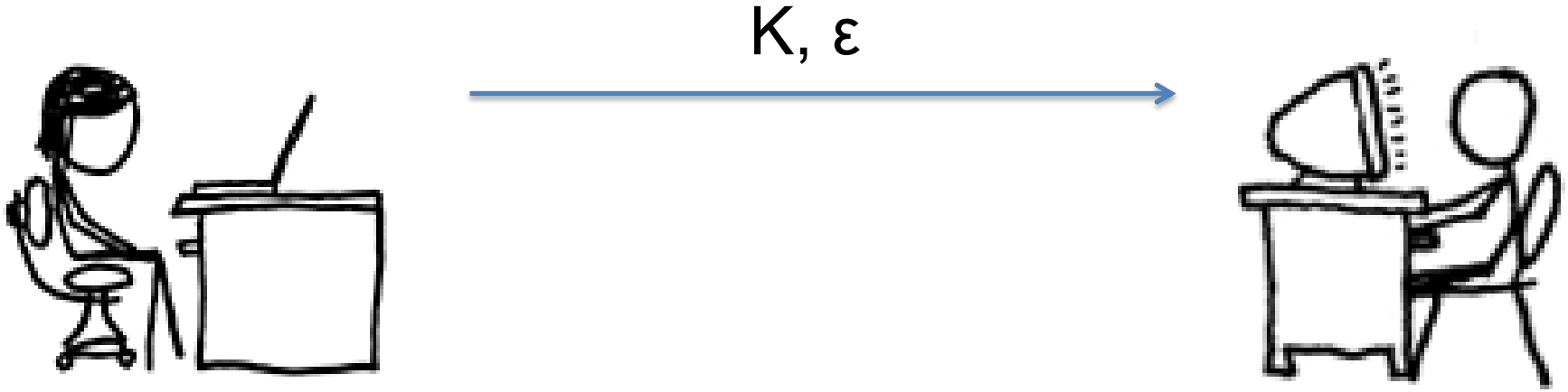
Proof of work

- Η Alice παράγει ένα τυχαίο αλάτι K, αρκετά μεγάλο ώστε να πιστεύει ότι δεν έχει επαναχρησιμοποιηθεί σε proof of work (π.χ. 128 bits).
- Στέλνει στον Bob το K και μία **παράμετρο στόχου** ε που ορίζει πόση δουλειά θέλει να κάνει

Proof of work

- Ο Bob υπολογίζει ένα x τέτοιο ώστε: $H(K||x) < \varepsilon$
- Το x αυτό ονομάζεται **nonce**
- Ο Bob στέλνει στην Alice το x
- Η Alice ελέγχει ότι $H(K||x) < \varepsilon$

Proof of work: Αίτηση



Proof of work: Εργασία

$H(K||x) < \varepsilon$?

Proof of work: Απόδειξη

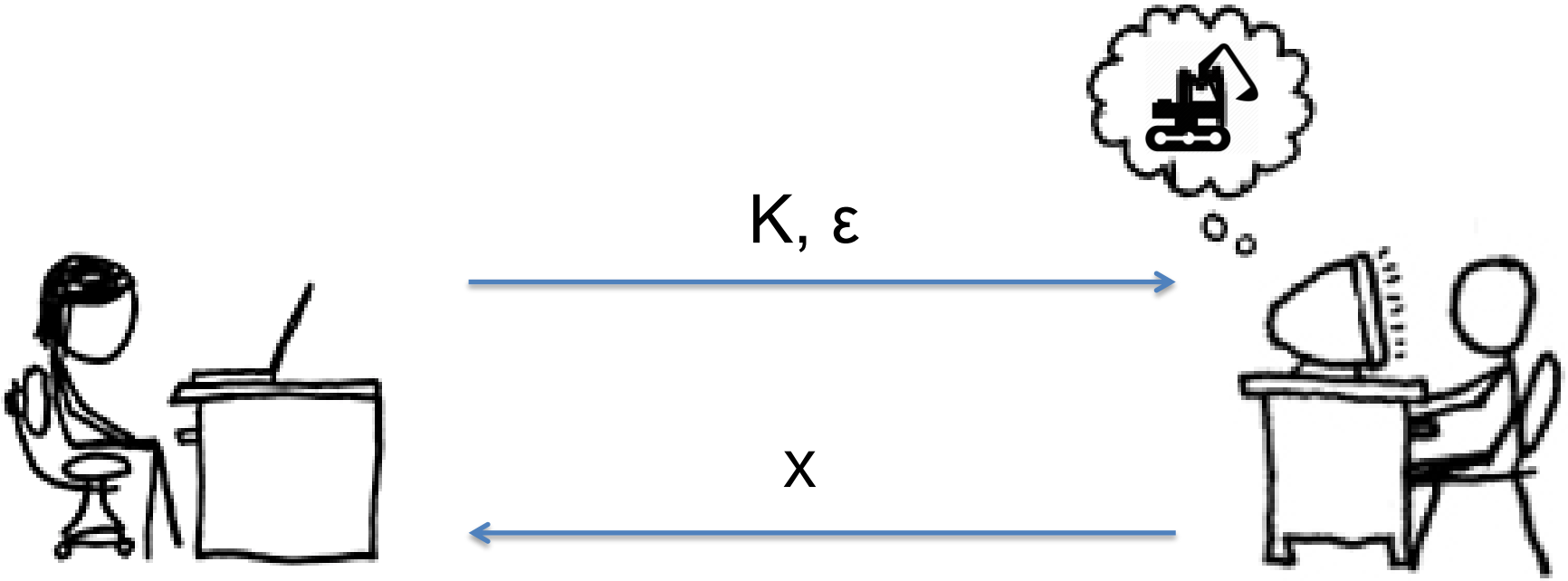
x

$H(K||x) < \varepsilon$?

Proof of work

```
do {
     x = rand();
} while (H(K||x) >= ε);
return x;
```

Proof of work



$H(K||x) < \varepsilon$?

Proof of work

- Χρησιμοποιώ hash function H με πεδίο τιμών $\{0, 1\}^n$
- Έστω X_i ανεξάρτητες τυχαίες μεταβλητές με ομοιόμορφη κατανομή στο $\{0, 1\}^m$, οι μεταβλητές preimage που δοκιμάζει ο Bob
- Έστω $Y_i = H(K||X_i)$
- Υποθέτουμε ότι Y_i **ανεξάρτητες** και ομοιόμορφα κατανεμημένες στο $\{0, 1\}^n$
- Ο Bob θέλει $Y_i < \varepsilon$ έτσι ώστε να πετύχει το Proof of work, έστω i = w όταν το πετυχαίνει.

Proof of work

```
do {
    x = rand();
} while (H(K||x) >= ε);
return x;
```

$X_i \sim \{0, 1\}^m$

$Y_i \sim \{0, 1\}^n$

Πόσες φορές (w) θα τρέξει η επανάληψη;

Proof of work

- Για δεδομένο i: $P(Y_i < \varepsilon) = \varepsilon / 2^n$
- Για $i \neq j$ και αρκετά μεγάλο m: $P(Y_i = Y_j) \approx 0$
- Y_i, Y_j ανεξάρτητες
- $P(w \leq \lambda) = \lambda\varepsilon / 2^n$
- $E(w) = 2^n / \varepsilon$

Proof of work

- Σημαντική εφαρμογή στο bitcoin
- Θα επανέλθουμε σε επόμενο μάθημα...

Merkle δέντρα

- Δομή δεδομένων που μας επιτρέπει να κάνουμε hashing σε πολλά μικρά data blocks
- Δυαδικό δέντρο
- Κάθε φύλλο είναι το hash κάποιων δεδομένων
- Ο γονιός είναι το hash των συνενωμένων παιδιών του
- Εφαρμογή σε BitTorrent, bitcoin, κρυπτογράφηση δίσκου...

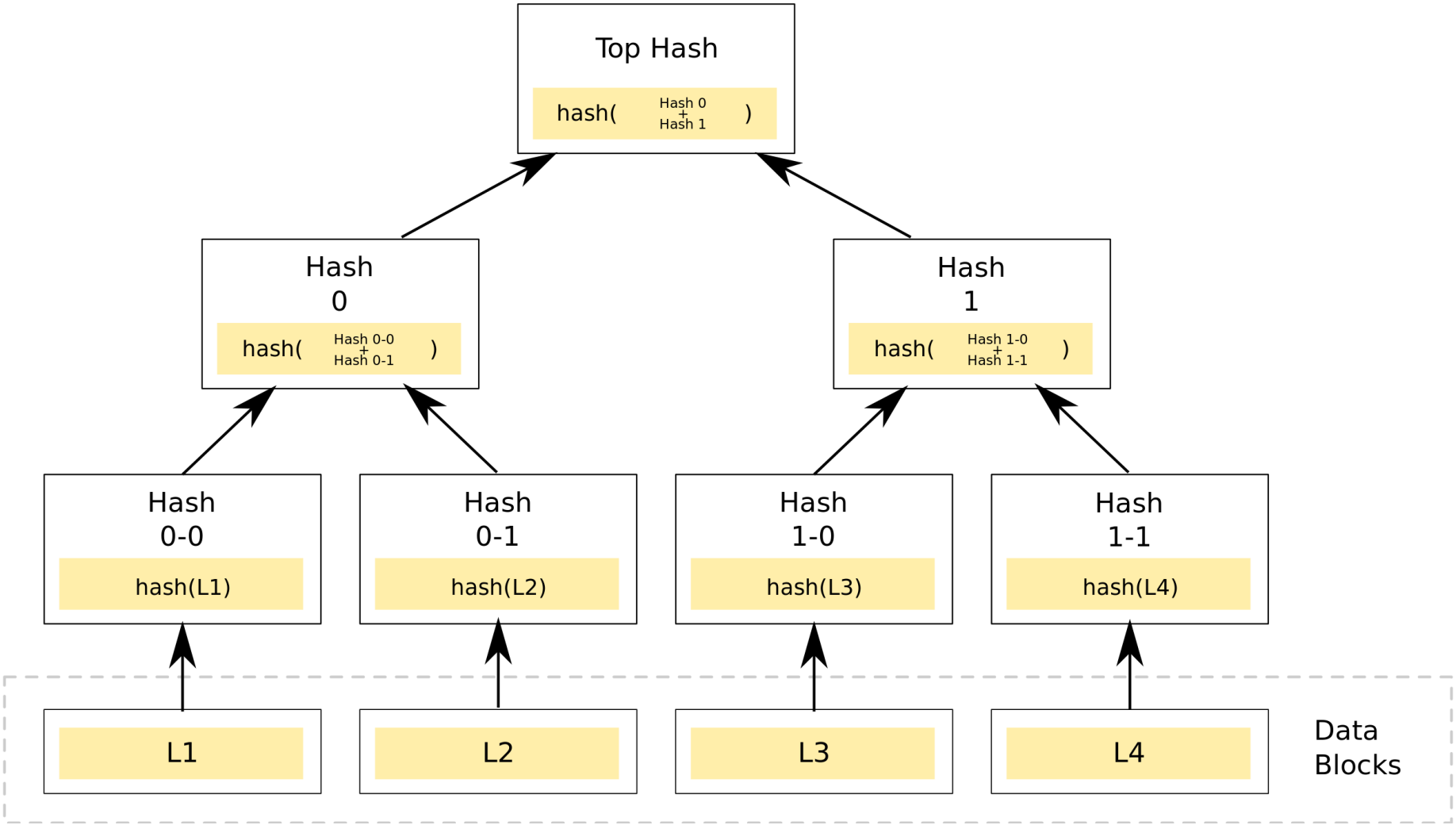
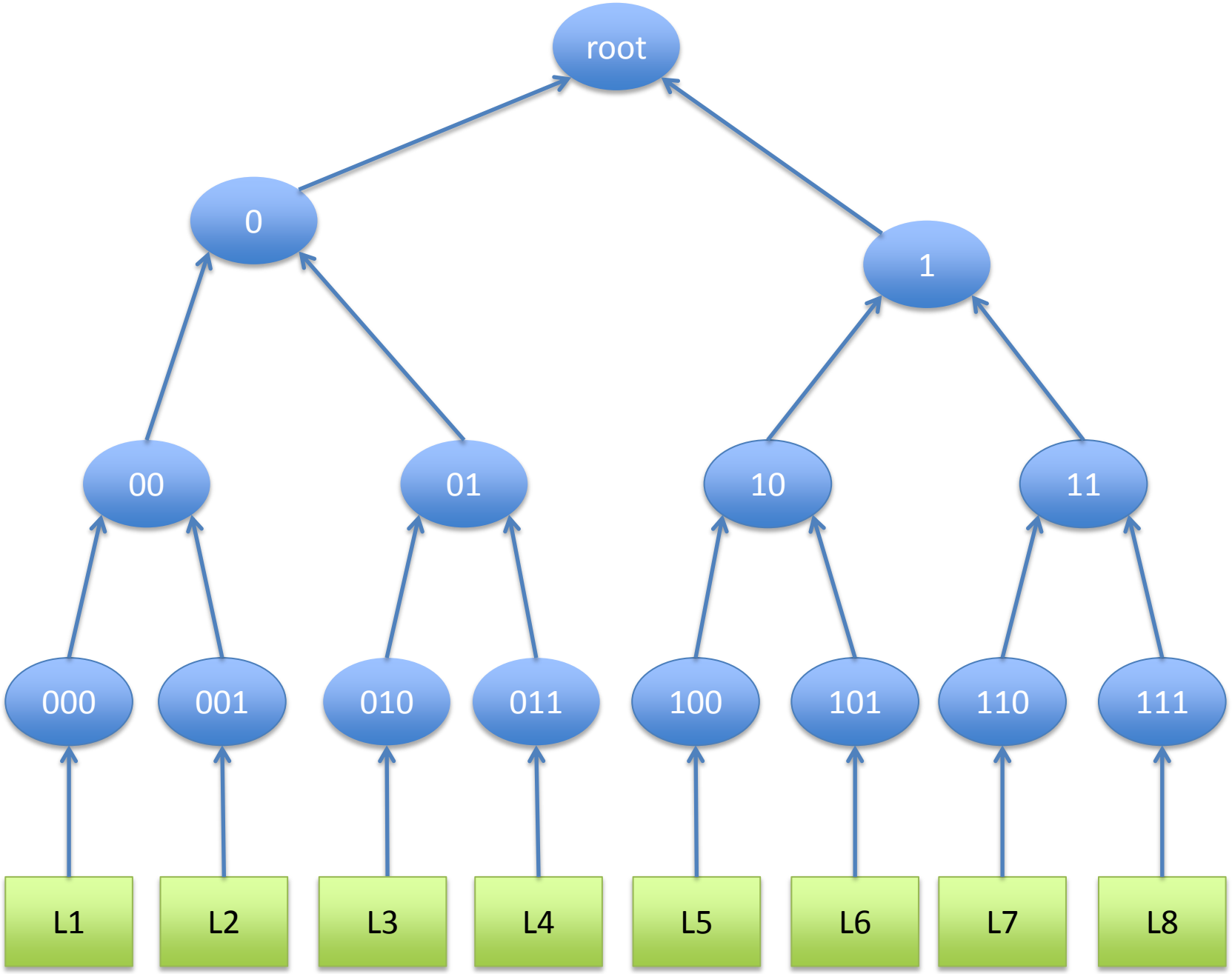


Image: David Göthberg


root
0
1
00
01
10
11
000
001
010
011
100
101
110
111
L1
L2
L3
L4
L5
L6
L7
L8

Merkle δέντρα

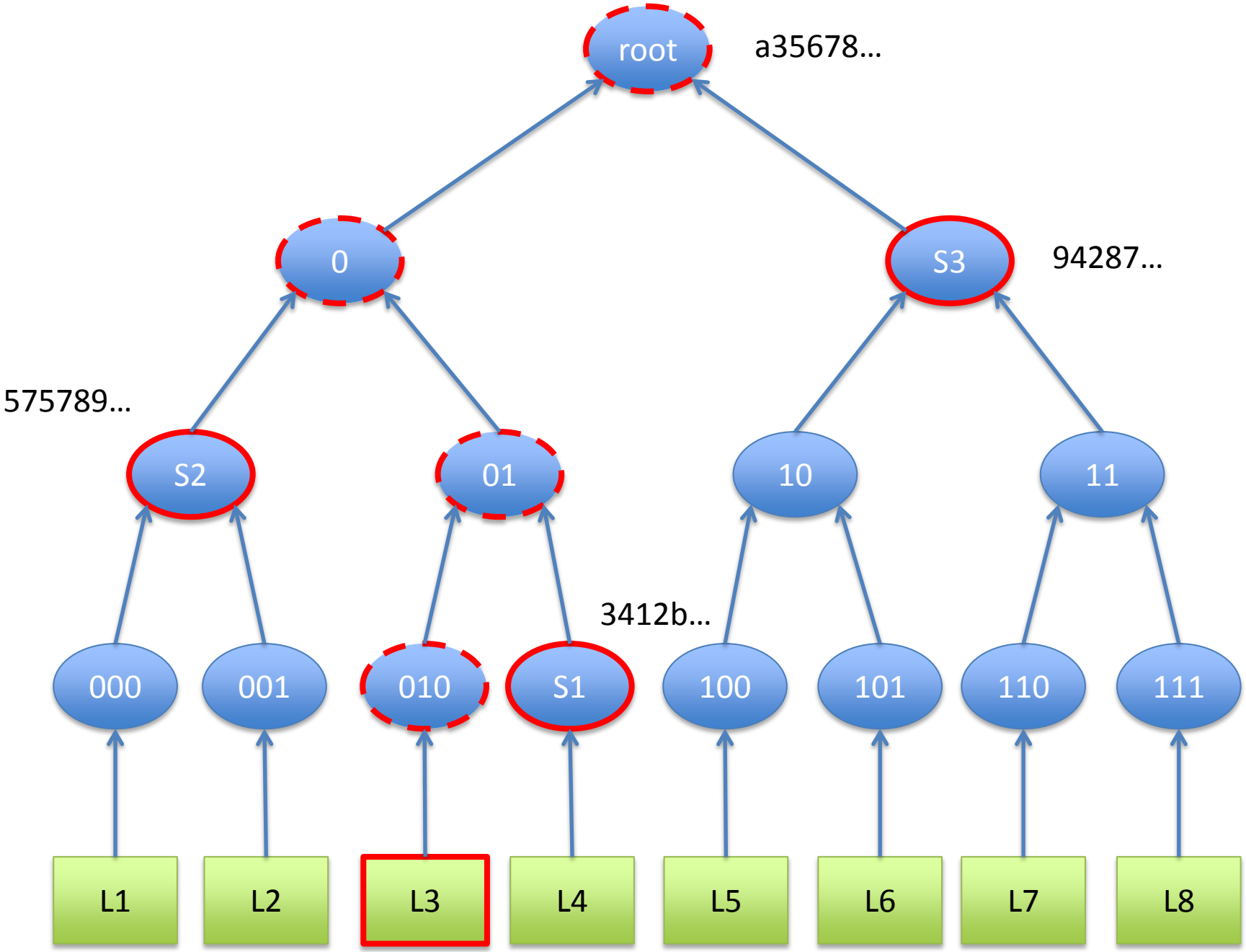
- Χρήσιμα για πιστοποίηση περιεχομένου
- Έστω ότι εμπιστευόμαστε πως η ρίζα είναι σωστή
- Τότε μπορεί κανείς να αποδείξει ότι ένα συγκεκριμένο περιεχόμενο είναι μέρος του δέντρου

Απόδειξη σε Merkle δέντρα

- Η Alice (Prover) θέλει να αποδείξει στον Bob (Verifier) ότι ένα συγκεκριμένο data block L έχει περιληφθεί σε ένα δέντρο
- Ο Bob γνωρίζει τη ρίζα του δέντρου
- Η Alice αποκαλύπτει στον Bob:
  - Το L
  - Τα hashes των αδερφών στο μονοπάτι που οδηγεί από το L στη ρίζα, S1, S2, S3, …, Sn
  - Το μονοπάτι (π.χ. αριστερά - δεξιά - δεξιά - δεξιά)

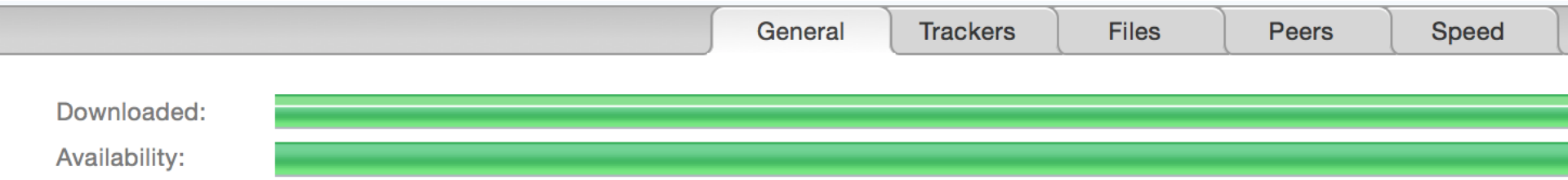
Απόδειξη σε Merkle δέντρα

- Ο Bob επιβεβαιώνει:
  - Υπολογίζει:
    - H(L)
    - H(H(L)||S1) - αριστερά
    - H(S2||H(H(L)||S1)) - δεξιά
    - κλπ.
  - Βρίσκει τελικά το hash της ρίζας
  - Ελέγχει αν είναι αυτό που ανέμενε
- Μέγεθος απόδειξης: Λογαριθμικό!


root
a35678...
0
S3
94287...
575789...
S2
01
10
11
3412b...
000
001
010
S1
100
101
110
111
L1
L2
L3
L4
L5
L6
L7
L8

Χρήση Merkle Trees στο BitTorrent

- Στο BitTorrent, ο κάθε χρήστης εμπιστεύεται τον tracker
- Αρχικά συνδέεται με τον tracker και κατεβάζει το merkle tree root
- Στη συνέχεια μπορεί να επιβεβαιώσει ότι οποιοδήποτε κομμάτι του torrent που κατεβάζει από άλλους είναι έγκυρο



TRANSFER

Time Elapsed:	4h 52m	Remaining:	
Downloaded:	2.9 GB	Uploaded:	439.0 MB
Download Speed:	0.0 kB/s (avg. 1.0 MB/s)	Upload Speed:	0.0 kB/s (avg. 25.7 kB/s)
Down Limit:	∞	Up Limit:	∞
Status:	Finished		

GENERAL

Save As:	/Users/dionyziz/Downloads/PPL Training		
Total Size:	7.26 GB (2.91 GB done)	Pieces:	1860 x 4.00 MB (have 745)
Created On:			
Hash:	b6a90ef2cfb3d14770bed7115e70787ee2a0e91f		
Comment:			

Μάθαμε

- Επεκτάσεις του collision resistance
- Δεσμεύσεις / hiding / binding
- Αλάτι
- md5, sha1, sha256, bcrypt, scrypt
- Proof of work
- Δέντρα Merkle
- Αποθήκευση κωδικών πρόσβασης
- Timestamping