



Οικονομικό Πανεπιστήμιο Αθηνών
Τμήμα Πληροφορικής
ΠΜΣ

Κρυπτογραφία και Εφαρμογές

Μαριάς Ιωάννης
marias@aueb.gr

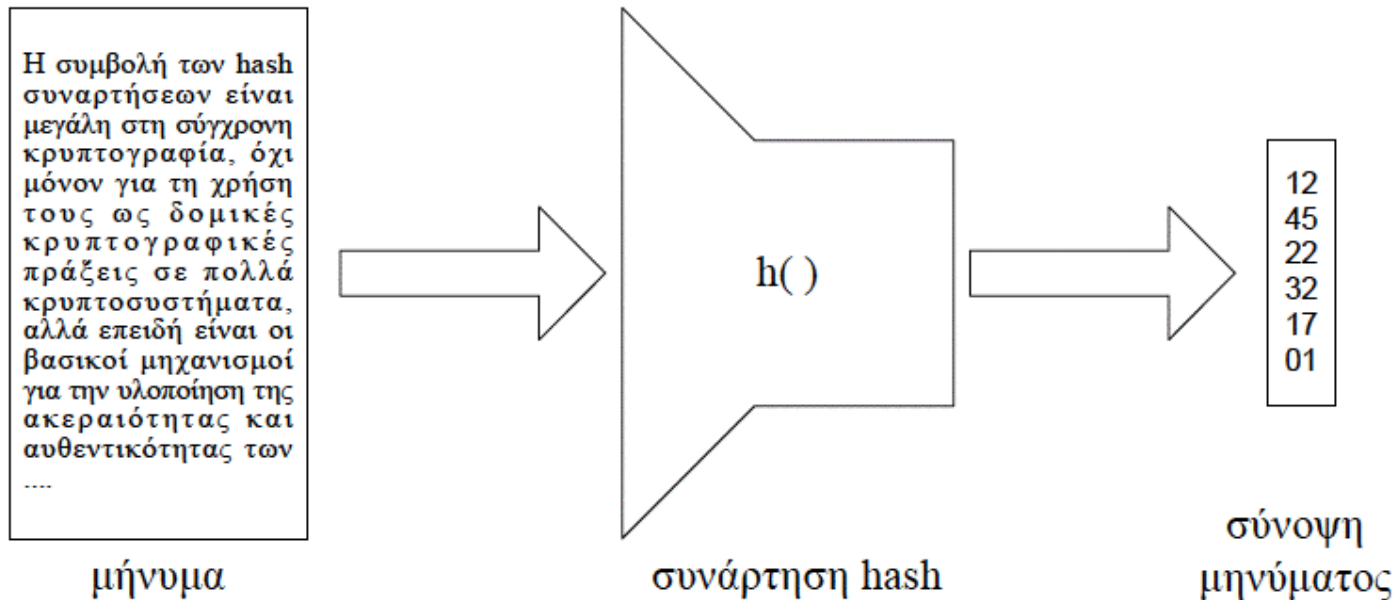
Μαρκάκης Ευάγγελος
markakis@gmail.com

- Hash functions (συναρτήσεις σύνοψης)
 - ✓ Assurance of Data Integrity (ακεραιότητα)
 - ✓ Προϋποθέσεις ασφάλειας
- Μέγεθος σύνοψης
- Message Authentication Codes (MACs)
- Modification Detection Codes (MDCs)
- Iterated Hash functions
- Υλοποιήσεις:
 - ✓ MD4, MD5 (Message Digest 5, Rivest)
 - ✓ SHA-1 (Secure Hash Algorithm)

- Αναφέρονται και ως *Συναρτήσεις Κατακερματισμού*
- Απεικονίζουν μήνυμα αυθαίρετου μήκους (*pre-image*) σε μήνυμα με σταθερό αριθμό bits
- Το αποτέλεσμα της διαδικασίας ονομάζεται *αποτύπωμα*, ή *hash value*, ή *message digest* ή *fingerprint*
- Γνωστοί αλγόριθμοι:
 - ✓ MD4, MD5 (Message Digest 5, Rivest)
 - ✓ SHA-1/2 (Secure Hash Algorithm)
 - ✓ RIPEMD 128/160 (RACE Integrity Primitives Evaluation Message Digest, Europe RACE Framework, 1996)

Λειτουργικές Προϋποθέσεις – Ευρύς Ορισμός

- ✓ Μπορεί να εφαρμοστεί σε τμήμα δεδομένων οποιουδήποτε μεγέθους
- ✓ Παράγει έξοδο (σύνοψη h) σταθερού **μικρού** μήκους
- ✓ Εύκολα υπολογίσιμη για δοθέν x



Προϋποθέσεις Ασφάλειας

Συνήθως θέλουμε μία συνάρτηση hash $h: X \rightarrow Y$ να ικανοποιεί τις ακόλουθες ιδιότητες:

1. Δεδομένης hash function h και $y \in Y$, πρέπει να είναι υπολογιστικά δύσκολο να βρεθεί x τέτοιο ώστε $h(x) = y$
(preimage resistance)
2. Δεδομένης hash function h και $x \in X$, πρέπει να είναι υπολογιστικά δύσκολο να βρεθεί x' τέτοιο ώστε $h(x') = h(x)$
(2nd preimage resistance ή weak collision resistance)
3. Δεδομένης hash function h , πρέπει να είναι υπολογιστικά δύσκολο να βρεθούν $x, x' \in X$ τέτοια ώστε $h(x) = h(x')$
(strong collision resistance)

Προϋποθέσεις Ασφάλειας

- Όταν ισχύει η ιδιότητα 1, θα λέμε ότι η h είναι one-way hash function (OWHF)
 - ✓ Η ύπαρξη one-way functions δεν έχει αποδειχθεί μαθηματικά
 - ✓ Στην πράξη όμως έχουμε αρκετές υποψήφιες συναρτήσεις που πιστεύουμε ότι είναι one-way
 - ✓ Η ασφάλεια σε πολλά κρυπτογραφικά σενάρια στηρίζεται στην ύπαρξη one-way functions
- Όταν ισχύει η ιδιότητα 3, θα λέμε ότι η h είναι collision resistant hash function (CRHF)

■ *Key-based hash functions*

- ✓ Μπορούμε να έχουμε και hash functions με κλειδί k ,
- ✓ **Παράδειγμα:** έστω μία συνάρτηση h χωρίς κλειδί
- ✓ Μπορούμε να ορίσουμε την h_k έτσι ώστε:

$$h_k(x) = h(x \parallel k)$$

όπου $x \parallel k$ το μήνυμα που προκύπτει από concatenation των x και k

- Πόσο μεγάλη πρέπει να είναι η σύνοψη σε μία collision resistant hash function?
- Από αρχή περιστερώννα σίγουρα θα υπάρχουν πολλές συγκρούσεις
- Θέλουμε να είναι υπολογιστικά ανέφικτο να βρίσκουμε συγκρούσεις με brute force.
- Παράδοξο Γενεθλίων: Σε μία ομάδα από 23 τυχαία επιλεγμένα άτομα, η πιθανότητα να υπάρχουν 2 άτομα που έχουν γενέθλια την ίδια μέρα είναι $\geq 1/2$

Απόδειξη: έστω n άτομα σε μία αίθουσα

- $P_Y(n)$: πιθανότητα τουλάχιστον δύο άτομα στο δωμάτιο να έχουν ίδια γενέθλια
- $P_N(n)$: πιθανότητα να μην υπάρχουν δύο άτομα στο δωμάτιο με ίδια γενέθλια.
- P_N και P_Y αλληλο-αποκλείονται: $P_Y(n) = 1 - P_N(n)$
- Ας θεωρήσουμε ότι τα άτομα έρχονται διαδοχικά στην αίθουσα

■ Παράδοξο Γενεθλίων

- ✓ Πιθανότητα 1^ο άτομο να μην έχει κοινά γενέθλια με υπόλοιπα άτομα στην αίθουσα = $P(1) = 365/365$
- ✓ Πιθ/τα 2^ο άτομο να μην έχει κοινά γενέθλια με 1^ο = $P(2) = 364/365$
- ✓ Πιθ/τα 3^ο άτομο να μην έχει κοινά γενέθλια με 1^ο και 2^ο = $P(3) = 363/365$
- ✓
- ✓ Πιθ/τα n-οστό άτομο να μην έχει κοινά γενέθλια με υπόλοιπους = $P(n) = (365-n+1)/365$

- ✓ Άρα $P_N(n) = P(1)*P(2)* \dots * P(n) = 365/365 \times 364/365 \times 363/365 \times 362/365 \times \dots \times (365-n+1)/365 =$

$$\frac{365!}{365^n \times (365-n)!}$$

■ Παράδοξο Γενεθλίων

✓ $P_Y(n) = 1 - P_N(n)$

- $n=10, P_Y(n)=11,7\%$
- $n=20, P_Y(n)=41,7\%$
- $n=23, P_Y(n)=50,7\%$
- $n=50, P_Y(n)=97,0\%$
- $n=100, P_Y(n)=99.99997\%$

✓ Προσέγγιση μέσω σειρών Taylor

- $P_Y(n) = 1 - e^{-n/n/(2*365)}$

- ✓ Ανάλογα αναλύουμε οποιαδήποτε άλλη κατάσταση όπου τα πιθανά ενδεχόμενα είναι $\neq 365$
- ✓ Το παράδοξο γενεθλίων πρέπει να λαμβάνεται υπόψη όταν αποφασίζουμε το μέγεθος της σύνοψης
- ✓ Δεν θέλουμε μεγάλη πιθ/τα 2 τυχαία μηνύματα να έχουν την ίδια σύνοψη
- ✓ Στην πράξη χρησιμοποιούνται συνήθως 160 bits
- ✓ Έχουν προταθεί και αλγόριθμοι για παραπάνω bits
- ✓ 128 μπορεί να είναι επίσης ok (έχουν βρεθεί επιθέσεις όμως)

- Παράδοξο Γενεθλίων – Αλγόριθμος Yuval
 - Αν η σύνοψη δεν είναι μεγάλη αυξάνεται η πιθ/τα «γειτονικά» κείμενα να έχουν το ίδιο hash
 - Μπορούμε τότε να προσπαθήσουμε να λύσουμε το εξής πρόβλημα:
 - ✓ Είσοδος: νόμιμο μήνυμα x_1 , Δόλιο μήνυμα x_2 , σύνοψη h μήκους m -bits
 - ✓ Έξοδος: x'_1, x'_2 αποτελέσματα ελάχιστων διαφοροποιήσεων των x_1 και x_2 , με $h(x'_1) = h(x'_2)$
1. Παραγωγή $t = 2^{m/2}$ κειμένων με ελάχιστες διαφοροποιήσεις x'_1 του x_1
 2. Για κάθε μήνυμα x'_1 παράγεται η σύνοψη και αποθηκεύεται μαζί με το αντίστοιχο μήνυμα ώστε να ανιχνεύονται με βάση τη hash-value (κόστος $O(t)$)
 3. Παράγονται κείμενα x'_2 με ελάχιστες διαφοροποιήσεις από το x_2 , υπολογίζονται τα $h(x'_2)$ για κάθε ένα κείμενο και συγκρίνονται με κάθε ένα από τα ως άνω x'_1 (μέχρι να βρεθεί ταίριασμα).

■ Παράδοξο Γενεθλίων – Αλγόριθμος Yuval

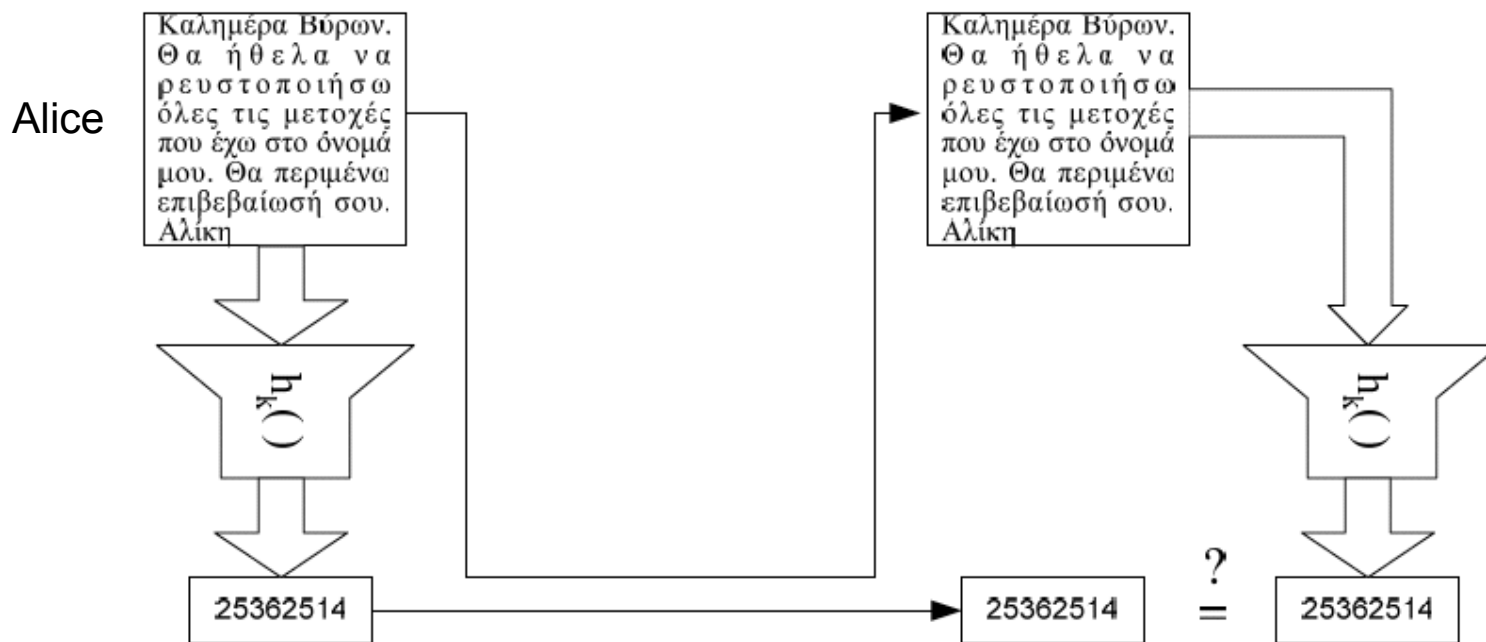
- ✓ Εφαρμογή αυτής της επίθεσης:
- ✓ Έστω ότι η Alice θέλει να υπογράψει μία συμφωνία x_1 με τον Bob
- ✓ Ο Bob δεν έχει ιδέα από κρυπτογραφία
- ✓ Η Alice παρουσιάζει μία παραλλαγή του αρχικού συμβολαίου x'_1 και ο Bob το υπογράφει
- ✓ Αργότερα η Αλίκη μπορεί να υποστηρίξει ότι το μήνυμα που υπεγράφη ήταν το x'_2

■ Εφαρμογές

- ✓ ψηφιακές υπογραφές
 - Το μεγάλο μήνυμα κατακερματίζεται και μόνο η hash τιμή υπογράφεται
 - εξοικονομεί χρόνο και χώρο σε σύγκριση με την υπογραφή του μηνύματος άμεσα
- ✓ ακεραιότητα των δεδομένων
 - Η hash-τιμή που αντιστοιχεί στα δεδομένα υπολογίζεται σε κάποια χρονική στιγμή
 - Η ακεραιότητα της hash-τιμής προστατεύεται
 - Σε μεταγενέστερη χρονική στιγμή επανα-υπολογίζεται η hash-τιμή των δεδομένων και γίνεται σύγκριση με την αρχική
- ✓ ακεραιότητα των δεδομένων και αυθεντικοποίηση πηγής δεδομένων (data origin authentication)
 - με key-based hash functions

- Οι hash functions που χρησιμοποιούνται για αυθεντικοποίηση και ακεραιότητα δεδομένων κατατάσσονται συνήθως σε:
 - ✓ Message Authentication Codes (MACs).
 - Key-based hash functions
 - Χρήση κυρίως για επικοινωνία μεταξύ 2 μελών όπου προέχει η ακεραιότητα δεδομένων αντί για εμπιστευτικότητα
 - Η Αλίκη στέλνει το μήνυμα μαζί με τη σύνοψη. Ο Bob όταν λάβει το μήνυμα, υπολογίζει ανεξάρτητα το MAC και ελέγχει αν είναι ίδιο με το MAC που έλαβε
 - ✓ Modification Detection Codes (MDCs):
 - Unkeyed hash functions
 - Χρήση κυρίως για αποστολή μηνύματος σε πολλούς παραλήπτες
 - Π.χ. Λήψη ηλεκτρονικών αγαθών, βιβλίων, software upgrades
 - Δεν υπάρχει ανάγκη για κλειδί, ο έλεγχος μπορεί να γίνει από οποιονδήποτε πελάτη

Message Authentication Codes (MACs).



- Παραδείγματα εφαρμογών MAC: σήματα πύργου ελέγχου αεροδρομίων, συντονισμός σιδηροδρομικών γραμμών
- Δεν απαιτείται εμπιστευτικότητα, αλλά μία αλλαγή του μηνύματος μπορεί να προκαλέσει ατυχήματα

■ Πιθανοί στόχοι αντιπάλου εναντίον MACs

- ✓ Εύρεση του κλειδιού k
- ✓ Χωρίς γνώση του k , εντοπισμός έγκυρου ζεύγους $(x, h_k(x))$
- ✓ Για δοσμένο x , εντοπισμός x' , έτσι ώστε $h_k(x') = h_k(x)$
- ✓ Εντοπισμός x, x' έτσι ώστε $h_k(x') = h_k(x)$

■ Στόχοι αντιπάλου εναντίον MDCs

- ✓ Τα ίδια εκτός του εντοπισμού κλειδιού

■ Επιθέσεις σε MAC/MDC

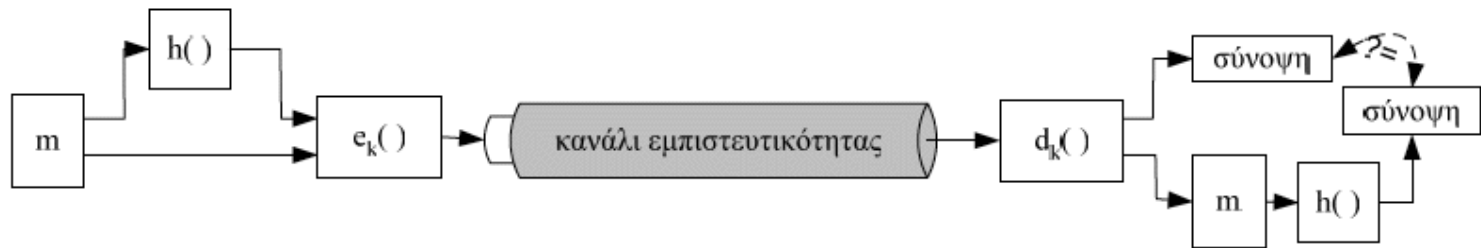
- ✓ *known-text attack*. Ένα η περισσότερα text-MAC pairs $(x_i, h_k(x_i))$ γίνονται γνωστά
- ✓ *chosen-text attack*. Ένα η περισσότερα text-MAC pairs $(x_i, h_k(x_i))$ γίνονται γνωστά για x_i που επιλέγονται από τον επιτιθέμενο
- ✓ *adaptive chosen-text attack*. Το x_i μπορεί να επιλέγεται από τον αντίπαλο, όπως παραπάνω, αλλά τώρα επιτρέπονται διαδοχικές επιλογές που βασίζονται στα αποτελέσματα των προηγούμενων

■ Διαφορές με κρυπταλγόριθμους

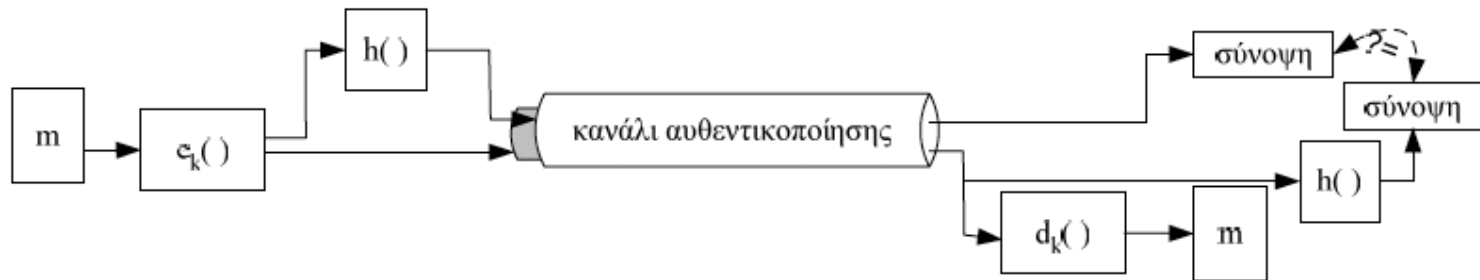
- ✓ Η συνάρτηση δεν χρειάζεται εδώ να είναι 1-1.
Περισσότεροι βαθμοί ελευθερίας
- ✓ Δεν υπάρχουν νομικοί περιορισμοί. Για κρυπταλγορίθμους μέγεθος κλειδιού νομικά ελεγχόμενο, εξαρτάται από τη χώρα (π.χ. στη Γαλλία είναι παράνομη η χρήση Vigenere αν η κυβέρνηση δεν έχει αντίγραφο κλειδιού)
- ✓ Ο αντίπαλος συνήθως γνωρίζει το κείμενο και τη σύνοψη (στους κρυπταλγορίθμους κάποιες φορές γνωρίζει μόνο το κρυπτοκείμενο)

■ Συνδυασμός εμπιστευτικότητας και αυθεντικοποίησης

- ✓ Υπάρχουν 2 επικρατέστερα σενάρια
- ✓ Κρυπτογράφηση της σύνοψης

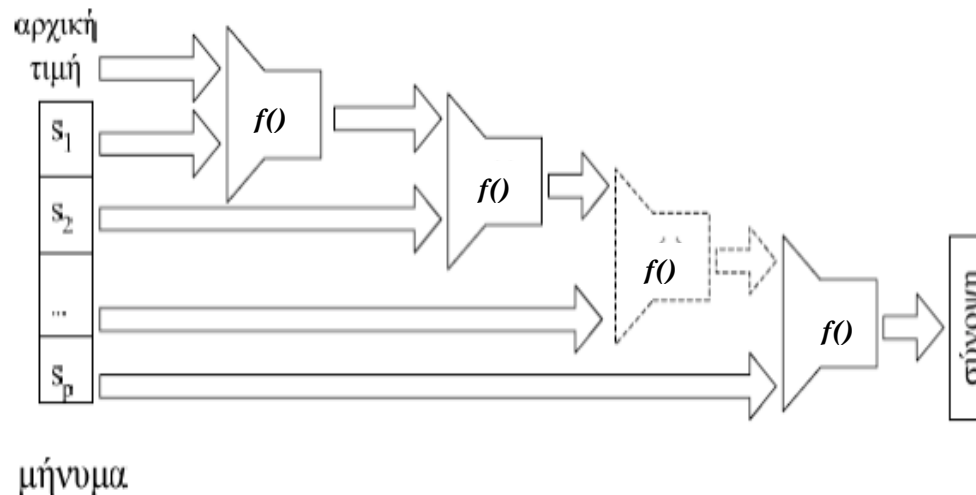


- ✓ Σύνοψη της κρυπτογράφησης



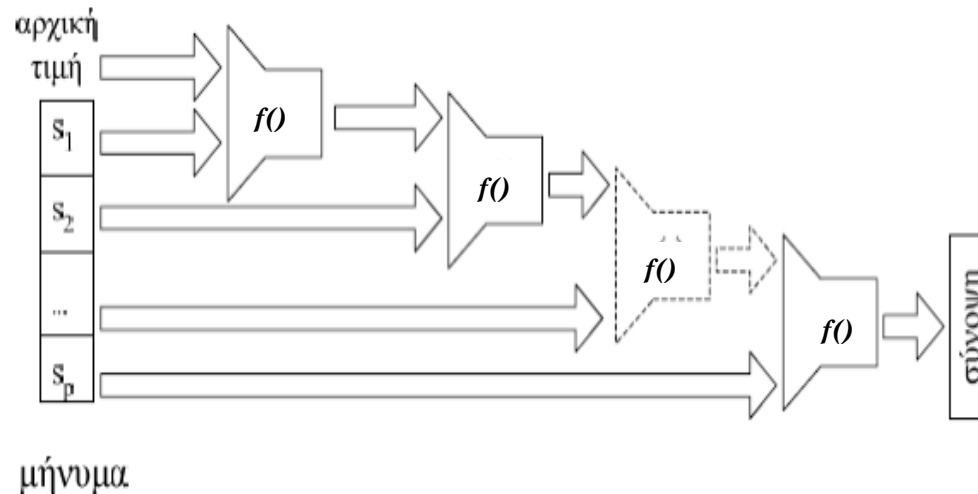
Iterated hash functions

- ✓ Είσοδος: μηνύματα με αυθαίρετο μήκος
- ✓ Έξοδος: σύνοψη σταθερού μήκους
- ✓ Κατάτμηση του μηνύματος της εισόδου σε τμήματα
- ✓ Επιμέρους επεξεργασία των τμημάτων αυτών.
- ✓ Βασίζονται στην επαναληπτική εφαρμογή μιας συνάρτησης f
- ✓ Η συνάρτηση $f: \{0, 1\}^m \rightarrow \{0, 1\}^n$ ονομάζεται **συνάρτηση συμπίεσης**



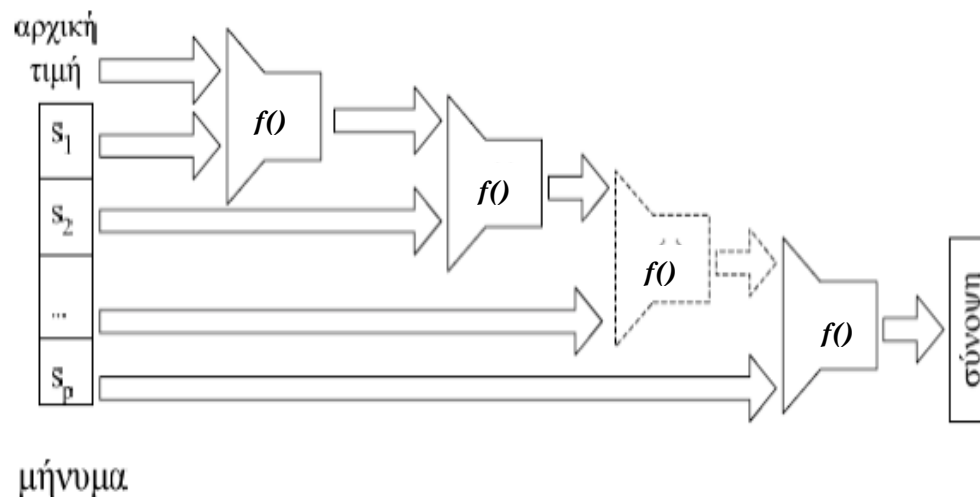
Iterated hash functions

- ✓ Συνήθως υπάρχουν 3 βήματα σε τέτοιες κατασκευές
- ✓ **Βήμα 1: Preprocessing (padding) step.** Από το input string x , μετά από κατάλληλο padding παράγεται ένα string $y = s_1 s_2 \dots s_p$. Κάθε τμήμα s_i έχει μήκος $m-n$ bits
- ✓ **Βήμα 2: Processing step.** Ξεκινάμε με ένα διάνυσμα αρχικοποίησης IV
 - $h_0 = IV$ (n bits)
 - $h_1 = f(h_0 || s_1)$
 - ...
 - $h_i = f(h_{i-1} || s_i), 1 \leq i \leq p$
 - Τελικά $h_p = f(h_{p-1}, s_p)$



Iterated hash functions

- ✓ Στο βήμα 2, κάθε block s_i είναι input της συμπίεσης f στην επανάληψη i
- ✓ Γίνεται concatenation με την έξοδο της προηγούμενης επανάληψης για να συμπληρωθεί το απαιτούμενο μήκος εισόδου των m bits.
- ✓ **Βήμα 3: Output Transformation (optional).** Προαιρετικά μία συνάρτηση g εφαρμόζεται στο αποτέλεσμα της τελευταίας επανάληψης
- ✓ Τελικά σε input x , το message digest είναι $h(x) = g(h_p)$



■ Iterated hash functions

- ✓ Η κατασκευή των Merkle και Damgard (1989)
- ✓ Έστω συνάρτηση συμπίεσης $f: \{0,1\}^{m+t} \rightarrow \{0,1\}^m$
- ✓ Έστω $|x|$ το μήκος του input string x (σε bits)

MERKLE-DAMGARD (x)

$n = |x|$

$k = \lceil n / (t-1) \rceil$ // χωρίζει το x σε τμήματα με μήκος $t-1$

$d = k(t-1) - n$ // πόσα bits θα γίνουν padded στο τελευταίο τμήμα

for $i=1$ to $k-1$ $y_i = x_i$

$y_k = x_k || 0^d$ //padding

$y_{k+1} = (\text{binary rep. of } d) || \text{padding}$

$z_1 = 0^{m+1} || y_1; h_1 = f(z_1)$

for $i=1$ to k //processing step

$z_{i+1} = h_i || 1 || y_{i+1}$

$h_{i+1} = f(z_{i+1})$

$h(x) = h_{k+1}$

■ Iterated hash functions

- ✓ Η κατασκευή θεωρεί ότι το αρχικό input string είναι στη μορφή $x = x_1 || x_2 || \dots || x_k$
- ✓ Επομένως το string $y = y_1 || y_2 || \dots || y_k || y_{k+1}$ είναι της μορφής $y = x || p(x)$, όπου $p(x)$ είναι το padding που καθορίζεται μόνο από το x
 - Είναι ο πιο συνηθισμένος τρόπος εκτέλεσης του preprocessing step
 - Το y_{k+1} πρέπει να γίνει padded με μηδενικά για να είναι και αυτό μήκους $t-1$
- ✓ **Θεώρημα:** Αν η συνάρτηση συμπίεσης f είναι collision-resistant τότε και η $h(x)$ όπως ορίζεται από την κατασκευή Merkle-Damgard θα είναι collision-resistant.

Iterated hash functions - History

- ✓ 1990: MD4, από τον Rivest
- ✓ 1992: MD5, πάλι από Rivest
- ✓ 1993: SHA ή SHA-0 (the Secure Hash Algorithm), προτυποποιείται ως FIPS-180
- ✓ 1995: SHA-1, FIPS 180-1
- ✓ mid-90's: collisions on MD4/MD5
- ✓ 1998: αλγόριθμος για εύρεση collisions στον SHA-0 σε 2^{61}
- ✓ 2002: FIPS 180-2. Εκτός από τον SHA-1, προτείνονται και οι SHA-224, SHA-256, SHA-384, SHA-512.
 - Η οικογένεια αυτή αναφέρεται και ως SHA-2
- ✓ 2004: Πιο ρεαλιστικός αλγόριθμος για εύρεση collisions σε SHA-0 και MD5 (CRYPTO 2004)
- ✓ 2005: Collisions για 58 γύρους του SHA-1, εκτιμάται ότι για τον κανονικό SHA-1, απαιτείται πολυπλοκότητα 2^{69}
 - Ανησυχία ότι στο μέλλον ο SHA-1 δεν θα είναι ασφαλής
- ✓ 2007:
 - Ο SHA-1 εξακολουθεί να είναι ο πιο δημοφιλής, lack of support για πλήρη χρήση του SHA-2
 - Ανακοινώνεται διαγωνισμός για SHA-3.
- ✓ 2012 (October 2): Ανακοινώνεται ο νικητής (Keccak)

■ MD5

- ✓ Ακολουθεί δομή Merkle-Damgård
- ✓ Το input string x μπορεί να είναι οποιουδήποτε μεγέθους
- ✓ Επιστρέφει hash τιμή 128 bits
- ✓ **Βήμα 1: Padding**

- Το x μετατρέπεται σε πολλαπλάσιο των 512 bits

$PAD(x)$

$$d = (447 - |x|) \bmod 512$$

$L = \text{bin. representation of } |x| // 64 \text{ bits, αν } |x| > 2^{64}, \text{ reduce mod } 2^{64}$

$$y = x || 1 || 0^d || L$$

- ✓ Το string y διαιρείται σε blocks των 512 bits
- ✓ $y = M_1 || M_2 || \dots || M_n$
- ✓ Κάθε M_i το βλέπουμε ως μία συλλογή 16 words (=32 bits)
- ✓ $M_i = W_0 || W_1 || \dots || W_{15}$

MD5

- ✓ **Βήμα 2:**
- ✓ Αρχικοποίηση βοηθητικών καταχωρητών
 - $H_0 = 67452301$
 - $H_1 = \text{EFCDAB89}$
 - $H_2 = 98\text{BADCFE}$
 - $H_3 = 10325476$
- ✓ Ενδιάμεσες τιμές αποθηκεύονται σε 4 καταχωρητές, A, B, C, D
- ✓ for $i = 1$ to n (όσα και τα blocks του x)
 - $A = H_0$
 - $B = H_1$
 - $C = H_2$
 - $D = H_3$
- ✓ Στη συνέχεια εκτελείται η συνάρτηση συμπίεσης: $f: \{0,1\}^{512+128} \rightarrow \{0,1\}^{128}$

MD5

- ✓ **Βήμα 2:**
- ✓ Συνάρτηση συμπίεσης: $f: \{0,1\}^{512+128} \rightarrow \{0,1\}^{128}$
- ✓ Αποτελείται από 4 γύρους
 - Κάθε γύρος j εκτελεί 1 πράξη f_j 16 φορές (με διαφορετικά ορίσματα)
 - Κάθε πράξη είναι μία μη γραμμική συνάρτηση πάνω σε 3 από τις μεταβλητές A, B, C, D και το αποτέλεσμα προστίθεται στην 4η μεταβλητή ($\text{mod } 2^{32}$)
- ✓ Γύρος j : $a = b + ((a + f_j(b, c, d) + W_t + T[k]) \lll s) \text{ mod } 2^{32}$
 - a, b, c, d : συνδυασμός των A, B, C, D . Η σειρά και αντιστοιχία μεταβάλλεται ανά γύρο και ανά πράξη
 - W_t : η t -οστή λέξη στο block που εξετάζουμε, $t=0, \dots, 15$
 - $T[k]$: ακέραιο τμήμα του $2^{32}|\sin(k)|$, $k = 1, \dots, 64$

MD5

- ✓ **Βήμα 2:**
- ✓ Οι πράξεις f_t
- ✓ $f_1(b, c, d) = (b \wedge c) \vee (\neg b \wedge d)$
- ✓ $f_2(b, c, d) = (b \wedge d) \vee (c \wedge \neg d)$
- ✓ $f_3(b, c, d) = b \oplus c \oplus d$
- ✓ $f_4(b, c, d) = c \oplus (b \vee \neg d)$

- ✓ Η τιμή για την κυκλική ολίσθηση διαφέρει σε κάθε επανάληψη

- ✓ Στο τέλος των 4 γύρων στις τελικές τιμές των A, B, C, D, προστίθενται οι αρχικές τιμές mod 2^{32}
 - $H_0 = H_0 + A$
 - $H_1 = H_1 + B$
 - $H_2 = H_2 + C$
 - $H_3 = H_3 + D$

- ✓ Επιστροφή στην αρχή του for loop για το επόμενο block του x

SHA-1

- ✓ Βασίζεται στον MD5
- ✓ Θεωρεί όπως και πριν ότι το μήκος της εισόδου δεν υπερβαίνει τα $2^{64}-1$ bits
- ✓ Παράγει συνοψη 160 bits
- ✓ **Βήμα 1:** το padding είναι ακριβώς ίδιο με MD5
- ✓ Στο τέλος του βήματος 1, έχουμε πάλι blocks των 512 bits (16 words)

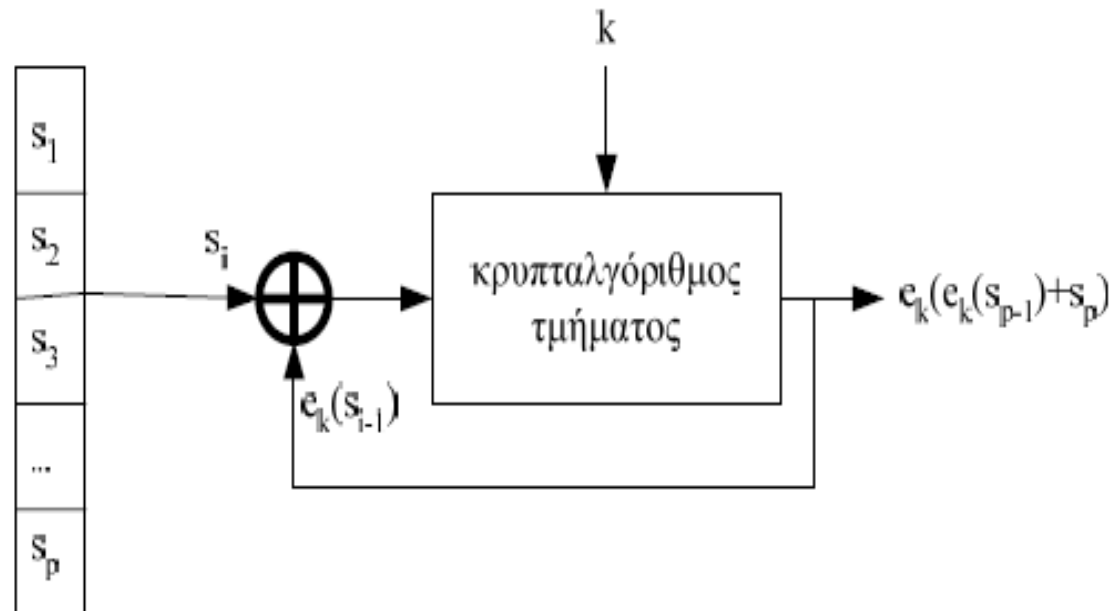
- ✓ **Βήμα 2:**
- ✓ Χρήση και 5ου καταχωρητή $E = C3D2E1F0$
- ✓ Αρχικές τιμές στους βοηθητικούς καταχωρητές H_0, H_1, H_2, H_3, H_4
- ✓ Η συνάρτηση συμπίεσης αποτελείται από 4 γύρους
- ✓ Κάθε γύρος j εκτελεί 20 επαναλήψεις εφαρμόζοντας μία πράξη f_j
- ✓ $f_1(b, c, d) = (b \wedge c) \vee (\neg b \wedge d)$
- ✓ $f_2(b, c, d) = b \oplus c \oplus d$
- ✓ $f_3(b, c, d) = (b \wedge c) \vee (b \wedge d) \vee (c \wedge d)$
- ✓ $f_4(b, c, d) = b \oplus c \oplus d$

SHA-1

- ✓ **Βήμα 2:**
- ✓ Πριν ξεκινήσουν οι 4 γύροι, το block των 16 words επεκτείνεται σε 80 words
 - Οι πρώτες 16 μένουν όπως είναι
 - For $t=16$ to 79 $W_t = (W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16}) \lll 1$
- ✓ Κάθε γύρος j εκτελεί 20 επαναλήψεις εφαρμόζοντας μία πράξη f_j
- ✓ $(a, b, c, d, e) = ((e+f_j(b,c,d) + S^5(A) + W_t + K_j), a, S^{30}(b), c, d)$
 - $S^k(\cdot)$: αριστερή κυκλική ολίσθηση κατά k bits
 - W_t : η t -οστή λέξη στο block που εξετάζουμε, $t=0, \dots, 79$
 - K_j : σταθερές τιμές για κάθε γύρο αποθηκευμένες σε 4 καταχωρητές
- ✓ Στο τέλος των 4 γύρων στις τελικές τιμές των A, B, C, D, E προστίθενται οι αρχικές τιμές
 - $H_0 = H_0 + A$
 - $H_1 = H_1 + B$
 - $H_2 = H_2 + C$
 - $H_3 = H_3 + D$
 - $H_4 = H_4 + E$
- ✓ Τελικά $SHA-1(x) = H_0 || H_1 || H_2 || H_3 || H_4$

■ Επαναληπτικές μονόδρομες hash συναρτήσεις

- ✓ Χρήση κρυπτοσυστήματος για δημιουργία hash functions
- ✓ Με κρυπταλγόριθμο τμήματος με n bits είσοδο και n bits έξοδο.
- ✓ Το ενδιάμεσο αποτέλεσμα προστίθεται στο επόμενο τμήμα του μηνύματος
- ✓ Η είσοδος του κλειδιού αποτελεί και το κλειδί της μονόδρομης hash



Επαναληπτική εφαρμογή hash function

- ✓ Αλυσίδα hash function – S/Key
 - Για παραγωγή One Time Password (used for one login session/transaction)
- ✓ Ξεκινά με secret key w .
 - Παράγεται από χρήστη ή αυτόματα
 - Αν αυτό παραβιαστεί τότε όλη η ασφάλεια του S/KEY σε κίνδυνο
 - Χρησιμοποιεί hash function H
 - Εφαρμόζεται n φορές στο w ,
 - Παράγει hash chain από n one-time passwords $p_1 \dots p_n$
 - Ο παραλήπτης εφοδιάζεται με τα n passwords με την αντίστροφη σειρά (δηλαδή p_n, \dots, p_2, p_1).

