CS306: Introduction to IT Security

Fall 2018

Lecture 9: Cloud & Network Security

Instructor: **Nikos Triandopoulos**

November 6, 2018

STEVENS
INSTITUTE *of* TECHNOLOGY
THE INNOVATION UNIVERSITY®

CS306: Tentative Syllabus

Week	Date	Topics	Reading	Assignment
1	Aug 28	Introduction	Ch. 1	-
2	Sep 4	Symmetric encryption	Ch. 2 & 12	Lab 1
3	Sep 11	Symmetric encryption II	Ch. 2 & 12	Lab 2, HW 1
4	Sep 18	Message authentication	Ch. 2 & 12	Lab 3, HW 1
5	Sep 25	Hash functions	Ch. 2 & 12	Lab 4
6	Oct 2	Public-key cryptography	Ch. 2 & 12	Lab 5
–	Oct 9	No class (Monday schedule)		Help session
7	Oct 16	Midterm (closed books)	All materials covered	No labs

CS306: Tentative Syllabus (continued)

Week	Date	Topics	Reading	Assignment
8	Oct 23	User authentication	Ch. 2	No labs
9	Oct 30	Access Control	Ch. 2	Lab 6
10	**Nov 6**	**Cloud & Network security**	**Ch. 6, 8**	**Lab 7, HW2, HW3**
11	Nov 13	Web, Software & Database		
12	Nov 20	Privacy		
13	Nov 27	Economics		
14	Dec 4	Legal & ethical issues		
15	Dec 11 (or later)	**Final** (closed books)	All materials covered*	

* w/ focus on what covered after midterm

Number-theoretic facts

Public-key encryption algorithms

Typically

- Based on **number theory & modular arithmetic**
- Security relies on some **hardness assumption**
  - a mathematical computational problem that is assumed to be hard to solve

We will learn about two such

- RSA
  - based on the hardness of factoring large numbers
- El Gamal
  - based on the hardness of solving discrete logarithm
  - using the same idea as Diffie-Hellman key agreement

Multiplicative inverses

The residues modulo a positive integer n comprise set $Z_n = \{0,1,2,\ldots,n - 1\}$

- let x and y be two elements in Z_n such that x y mod n = 1
  - we say: y is the multiplicative inverse of x in Z_n
  - we write: $y = x^{-1}$
- example:
  - multiplicative inverses of the residues modulo 11

x	0	1	2	3	4	5	6	7	8	9	10
x^{-1}		1	6	4	3	9	2	8	7	5	10

Multiplicative inverses (cont'ed)

Fact

An element x in Z_n has a multiplicative inverse iff x and n are **relatively prime**

- e.g., the only elements of Z_{10} having a multiplicative inverse are 1, 3, 7, 9

x	0	1	2	3	4	5	6	7	8	9
x^{-1}		1		7				3		9

Corollary

If p is prime, every nonzero residue in Z_p has a multiplicative inverse

Fact

If they exist, inverses are computed by the **Extended Euclidean Algorithm**

Multiplicative group Z_n^*

Defined as the subset of Z_n containing all integers that are relative prime to n

- closed under multiplication, every element has an inverse

Two cases

- if n is a **prime** number, then all non-zero elements in Z_n have an inverse
  - $Z^*_7 = \{1,2,3,4,5,6\}$, n = 7
  - $2 \bullet 4 = 1 \pmod 7$, $3 \bullet 5 = 1 \pmod 7$, $6 \bullet 6 = 1 \pmod 7$, $1 \bullet 1 = 1 \pmod 7$
- if n is **not prime**, not all integers in Z_n have an inverse
  - $Z^*_{10} = \{1,3,7,9\}$, n = 10
  - $3 \bullet 7 = 1 \pmod{10}$, $9 \bullet 9 = 1 \pmod{10}$, $1 \bullet 1 = 1 \pmod{10}$

Order of a multiplicative group

The order of a group is its cardinality of group (i.e., number of its elements)

- the totient function $\phi(n)$ denotes the order of Z^*_n , i.e., $\phi(n) = |Z^*_n|$
- **if n = p is prime**, then the order of $Z^*_p=\{1,2,\ldots,p-1\}$ is p-1, i.e., $\phi(n) = p-1$
  - e.g., $Z^*_7 = \{1,2,3,4,5,6\}$, $n = 7$, $\phi(7) = 6$
- **if n is not prime**, $\phi(n) = n(1-1/p_1)(1-1/p_2)\ldots(1-1/p_k)$, where $n = p^{e1}_1 p^{e2}_2 \ldots p^{ek}_k$
  - e.g., $Z^*_{10} = \{1,3,7,9\}$, $n = 10$, $\phi(10) = 4$
- if n = p q, where p and q are distinct primes, then $\phi(n) = (p-1)(q-1)$
  - difficult problem: Given n = pq, where p, q are primes, find p and q or $\phi(n)$

Fermat's Little Theorem (**case 1**)

Theorem

If p is a prime, then for each nonzero x in Z_p, we have $x^{p-1} \bmod p = 1$

- example (p = 5):

  $1^4 \bmod 5 = 1$

  $2^4 \bmod 5 = 16 \bmod 5 = 1$

  $3^4 \bmod 5 = 81 \bmod 5 = 1$

  $4^4 \bmod 5 = 256 \bmod 5 = 1$

Corollary

If p is a prime, then the multiplicative inverse of each nonzero residue x in Z_p is $x^{p-2} \bmod p$

- proof:

  $x(x^{p-2} \bmod p) \bmod p = xx^{p-2} \bmod p = x^{p-1} \bmod p = 1$

Euler's Theorem (**case 2**)

Theorem

For each element x in Z^*_n, we have $x^{\phi(n)} \bmod n = 1$

- example (n = 10)
  - $Z^*_{10} = \{1,3,7,9\}$, n = 10, $\phi(10) = 4$
  - $3^{\phi(10)} \bmod 10 = 3^4 \bmod 10 = 81 \bmod 10 = 1$
  - $7^{\phi(10)} \bmod 10 = 7^4 \bmod 10 = 2401 \bmod 10 = 1$
  - $9^{\phi(10)} \bmod 10 = 9^4 \bmod 10 = 6561 \bmod 10 = 1$

Computing in the exponent

For the multiplicative group Z^*_n, we can reduce the exponent modulo $\phi(n)$

$$x^y \bmod n = x^{k\,\phi(n)+r} \bmod n = (x^{\phi(n)})^k x^r \bmod n = x^{\,y \bmod \phi(n)} \bmod n$$

- **example**
  - $Z^*_{10} = \{1,3,7,9\}$, $n = 10$, $\phi(10) = 4$
  - $3^{1590} \bmod 10 = 3^{1590 \bmod 4} \bmod 10 = 3^2 \bmod 10 = 9$
  - how about $2^8 \bmod 10$?

Corollary: For Z^*_p, we can reduce the exponent modulo p-1

- **example**
  - $Z^*_p = \{1,2,\ldots,p-1\}$, $p = 19$, $\phi(19) = 18$
  - $15^{39} \bmod 19 = 15^{39 \bmod 18} \bmod 19 = 15^3 \bmod 19 = 12$

Euclid's GCD algorithm

Computes the greater common divisor by repeatedly applying the formula

gcd(a, b) = gcd(b, a mod b)

- example
  - gcd(412, 260) = 4

```
Algorithm EuclidGCD(a, b)
    Input integers a and b
    Output gcd(a, b)

    if b = 0
        return a
    else
        return EuclidGCD(b, a mod b)
```

a	412	260	152	108	44	20	4
b	260	152	108	44	20	4	0

*Proof of correctness

We need to prove that GCD(**a**, **b**) = GCD(**b**, **a** mod **b**)

- facts
  - every divisor of **a** and **b** is a divisor of **b** and (**a** mod **b**)
    - because (**a** mod **b**) can be written as the sum of **a** and a multiple of **b**, i.e., **a** mod **b** = **a** + k**b**, for some integer k
  - similarly, every divisor of **b** and (**a** mod **b**) is a divisor of **a** and **b**
    - because **a** can be written as the sum of (**a** mod **b**) and a multiple of **b**, i.e., **a** = k**b** + (**a** mod **b**), for some integer k
  - therefore the set of all divisors of **a** and **b** is **the same** with the set of all divisors of **b** and (**a** mod **b**), thus the greatest should also be the same

Extended Euclidean algorithm

Theorem

If, given positive integers ***a*** and ***b***, ***d*** is the smallest positive integer s.t. ***d*** = ***ia*** + ***jb***, for some integers ***i*** and ***j***, then ***d*** = gcd(***a, b***)

- example
  - ***a*** = 21, ***b*** = 15
  - ***d*** = 3, ***i*** = 3, ***j*** = -4
  - 3 = 3·21 + (-4)·15 = 63 - 60 = 3

Algorithm ***Extended-Euclid***(***a, b***)
  Input integers ***a*** and ***b***
  Output gcd(***a***, ***b***), *i* and *j*
    s.t. ia+jb = gcd(a,b)
  if ***b*** = 0
    return (a,1,0)
  (d', x', y') = ***Extended-Euclid***(***b, a*** mod ***b***)
  (d, x, y) = (d', y', x' - [a/b]y')
  return (d, x, y)

Computing multiplicative inverses

Goal: Compute the multiplicative inverse of a in Z_b

Fact

- given two numbers **a** and **b**, there exist integers x, y s.t. **x a + y b = gcd(a,b)**
- can be computed efficiently by the extended Euclidean algorithm

Therefore

- the multiplicative inverse of a in Z_b exists if and only if gcd(a, b) = 1
  - if the extended Euclidean algorithm computes x and y s.t. **x a + y b = 1** then the multiplicative inverse of a in Z_b is **x**

The RSA algorithm

The RSA algorithm (for encryption)

General case

Setup (run by a given user)

- $\mathbf{n} = \mathbf{p} \cdot \mathbf{q}$, with **p** and **q** primes
- **e** relatively prime to $\phi(n) = (\mathbf{p} - 1)(\mathbf{q} - 1)$
- **d** inverse of **e** in $\mathbf{Z}_{\phi(n)}$

Keys

- public key is $\mathbf{K}_{PK} = (\mathbf{n}, \mathbf{e})$
- private key is $\mathbf{K}_{SK} = \mathbf{d}$

Encryption

- $\mathbf{C} = \mathbf{M}^{\mathbf{e}} \bmod \mathbf{n}$ for plaintext **M** in $\mathbf{Z}_{\mathbf{n}}$

Decryption

- $\mathbf{M} = \mathbf{C}^{\mathbf{d}} \bmod \mathbf{n}$

Example

Setup

- $\mathbf{p} = 7$, $\mathbf{q} = 17$, $\mathbf{n} = 7 \cdot 17 = 119$
- $\mathbf{e} = 5$, $\phi(\mathbf{n}) = 6 \cdot 16 = 96$
- $\mathbf{d} = 77$

Keys

- public key is (119, 5)
- private key is 77

Encryption

- $\mathbf{C} = 19^5 \bmod 119 = 66$ for $\mathbf{M} = 19$ in $\mathbf{Z}_{119}$

Decryption

- $\mathbf{M} = 66^{77} \bmod 119 = 19$

Another complete example

- Setup
  - **p** = 5, **q** = 11, **n** = 5 · 11 = 55
  - **ϕ**(**n**) = 4 · 10 = 40
  - **e** = 3, **d** = 27 (3·27 = 81 = 2·40 + 1)
- Encryption
  - **C** = $\mathbf{M}^3$ mod 55 for **M** in $\mathbf{Z}_{55}$
- Decryption
  - **M** = $\mathbf{C}^{27}$ mod 55

M	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
C	1	8	27	9	15	51	13	17	14	10	11	23	52	49	20	26	18	2
M	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36
C	39	25	21	33	12	19	5	31	48	7	24	50	36	43	22	34	30	16
M	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54
C	53	37	29	35	6	3	32	44	45	41	38	42	4	40	46	28	47	54

*Correctness of RSA

Given

Setup

- **n** = **p** · **q**, with **p** and **q** primes
- **e** relatively prime to **ϕ**(n) = (**p** - 1)(**q** - 1)
- **d** inverse of **e** in $\mathbf{Z}_{\phi(n)}$ **(1)**

Encryption

- **C** = $\mathbf{M}^{\mathbf{e}}$ mod **n** for plaintext **M** in $\mathbf{Z}_{\mathbf{n}}$

Decryption

- **M** = $\mathbf{C}^{\mathbf{d}}$ mod **n**

Fermat's Little Theorem **(2)**

- for prime p, non-zero x: x^{p-1} mod p = 1

Analysis

Need to show

- M^{ed} = M mod p · q

Use **(1)** and apply **(2)** for prime p

- $M^{ed} = M^{ed-1} M = (M^{p-1})^{h(q-1)} M$
- $\mathbf{M^{ed}} = 1^{h(q-1)} M$ mod p = **M mod p**

Similarly (w.r.t. prime q)

- $\mathbf{M^{ed}}$ **= M mod q**

Thus, since p, q are co-primes

- $\mathbf{M^{ed}}$ **= M mod p · q**

A useful symmetry

[1] RSA setting

- modulo **n** = **p** · **q**, p & q are **primes**, public & private keys (e,d): **d** · **e** = **1 mod (p-1)(q-1)**

[2] RSA operations involve **exponentiations**, thus they are **interchangeable**

- **C** = **M^e** mod **n** (encryption of plaintext **M in Z_n**)
- **M** = **C^d** mod **n** (decryption of ciphertext **C in Z_n**)

Indeed, their order of execution does not matter: **$(M^e)^d = (M^d)^e$ mod n**

[3] RSA operations involve exponents that **"cancel out",** thus they are **complementary**

- **$x^{(p-1)(q-1)}$** mod **n** = **1** (Euler's Theorem)

Indeed, they invert each other: **$(M^e)^d = (M^d)^e = M^{ed} = M^{k(p-1)(q-1)+1}$ mod n**

$= (M^{(p-1)(q-1)})^k \cdot M = 1^k \cdot M = M$ mod n

Signing with RSA

RSA functions are complementary & interchangeable w.r.t. order of execution

- **core property**: $\mathbf{M^{ed} = M \bmod p \cdot q}$ for any message **M in** $\mathbf{Z_n}$

RSA cryptosystem lends itself to a **signature scheme**

- 'reverse' use of keys is possible : $\mathbf{(M^d)^e = M} \bmod \mathbf{p \cdot q}$
- signing algorithm **Sign(M,d,n)**: $\boldsymbol{\sigma} = \mathbf{M^d} \bmod \mathbf{n}$ for message **M** in $\mathbf{Z_n}$
- verifying algorithm **Vrfy(σ,M,e,n)**: return **M** == $\boldsymbol{\sigma}^{\mathbf{e}} \bmod \mathbf{n}$

The RSA algorithm (for signing)

General case

Setup (run by a given user)

- $\mathbf{n} = \mathbf{p} \cdot \mathbf{q}$, with **p** and **q** primes
- **e** relatively prime to $\phi(n) = (\mathbf{p} - 1)(\mathbf{q} - 1)$
- **d** inverse of **e** in $\mathbf{Z}_{\phi(n)}$

Keys (same as in encryption)

- public key is $\mathbf{K}_{PK} = (\mathbf{n}, \mathbf{e})$
- private key is $\mathbf{K}_{SK} = \mathbf{d}$

Sign

- $\sigma = \mathbf{M}^{\mathbf{d}} \bmod \mathbf{n}$ for message **M** in $\mathbf{Z}_{\mathbf{n}}$

Verify

- Check if $\mathbf{M} = \sigma^{\mathbf{e}} \bmod \mathbf{n}$

Example

Setup

- $\mathbf{p} = 7$, $\mathbf{q} = 17$, $\mathbf{n} = 7 \cdot 17 = 119$
- $\mathbf{e} = 5$, $\phi(\mathbf{n}) = 6 \cdot 16 = 96$
- $\mathbf{d} = 77$

Keys

- public key is (119, 5)
- private key is 77

Signing

- $\sigma = 66^{77} \bmod 119 = 19$ for $\mathbf{M} = 66$ in $\mathbf{Z}_{119}$

Verification

- Check if $\mathbf{M} = 19^5 \bmod 119 = 66$

Digital signatures & hashing

Very often digital signatures are used with hash functions

- the hash of a message is signed, instead of the message itself

Signing message M

- let h be a cryptographic hash function, assume RSA setting (n, d, e)
- compute signature σ on message M as: $\sigma = h(M)^d \bmod n$
- send σ, M

Verifying signature σ

- use public key (e, n) to compute (candidate) hash value $H = \sigma^e \bmod n$
- if H = h(M) output ACCEPT, else output REJECT

Security of RSA

Based on difficulty of **factoring** large numbers (into large primes), i.e., $n = p \cdot q$ into p, q

- note that for RSA to be secure, both p and q must be large primes
- widely believed to hold true
  - since 1978, subject of extensive cryptanalysis without any serious flaws found
  - best known algorithm takes exponential time in security parameter (key length $|n|$)
- how can you break RSA if you can factor?

Current practice is using 2,048-bit long RSA keys (617 decimal digits)

- estimated computing/memory resources needed to factor an RSA number within one year

Length (bits)	PCs	Memory
430	1	128MB
760	215,000	4GB
1,020	342×10^6	170GB
1,620	1.6×10^{15}	120TB

RSA challenges

Challenges for breaking the RSA cryptosystem of various key lengths (i.e., |n|)

- known in the form RSA-`key bit length' expressed in bits or decimal digits
- provide empirical evidence/confidence on strength of specific RSA instantiations

Known attacks

- RSA-155 (**512-bit**) factored in **4 mo**. using 35.7 CPU-years or 8000 Mips-years (**1999**) and 292 machines
  - 160 175-400MHz SGI/Sun, 8 250MHz SGI/Origin, 120 300-450MHz Pent. II, 4 500MHz Digital/Compaq
- RSA-**640** factored in **5 mo**. using 30 2.2GHz CPU-years (**2005**)
- RSA-220 (**729-bit**) factored in **5 mo**. using 30 2.2GHz CPU-years (**2005**)
- RSA-232 (**768-bit**) factored in **2 years** using **parallel** computers 2K CPU-years (1-core 2.2GHz AMD Opteron) (**2009**)

Most interesting challenges

- prizes for factoring RSA-**1024**, RSA-**2048** is $100K, $200K – estimated at 800K, 20B Mips-centuries

Deriving an RSA key pair

- public key is pair of integers (e,n), secret key is (d, n) or d
- the value of n should be quite large, a product of two large primes, p and q
- often p, q are nearly 100 digits each, so n ~= 200 decimal digits (~512 bits)
  - but 2048-bit keys are becoming a standard requirement nowadays
- the larger the value of n the harder to factor to infer p and q
  - but also the slower to process messages
- a relatively large integer e is chosen
  - e.g., by choosing e as a prime that is larger than both $(p - 1)$ and $(q - 1)$
  - why?
- d is chosen s.t. $e \cdot d = 1 \bmod (p - 1)(q - 1)$
  - how?

Discussion on RSA

- Assume **p** = 5, **q** = 11, **n** = 5 · 11 = 55, **ϕ**(**n**) = 40, **e** = 3, **d** = 27
  - why encrypting small messages, e.g., **M** = 2, 3, 4 is tricky?
  - recall that the ciphertext is **C** = $\mathbf{M}^3$ mod 55 for **M** in $\mathbf{Z}_{55}$

M	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
C	1	8	27	9	15	51	13	17	14	10	11	23	52	49	20	26	18	2
M	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36
C	39	25	21	33	12	19	5	31	48	7	24	50	36	43	22	34	30	16
M	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54
C	53	37	29	35	6	3	32	44	45	41	38	42	4	40	46	28	47	54

Discussion on RSA

- Assume **p** = 5, **q** = 11, **n** = 5 · 11 = 55, **ϕ**(**n**) = 40, **e** = 3, **d** = 27
  - why encrypting small messages, e.g., **M** = 2, 3, 4 is tricky?
  - recall that the ciphertext is **C** = $\mathbf{M}^3$ mod 55 for **M** in $\mathbf{Z}_{55}$
- Assume n = 20434394384355534343545428943483434356091 = p · q
  - can e be the number 4343253453434536?
- Are there problems with applying RSA in practice?
  - what other algorithms are required to be available to the user?
- Are there problem with respect to RSA security?
  - does it satisfy CPA security?

Algorithmic issues

The implementation of the RSA cryptosystem requires various algorithms

- Main issues
  - representation of integers of arbitrarily large size; and
  - arithmetic operations on them, namely computing modular powers
- Required algorithms (at setup)
  - generation of **random numbers** of a given number of bits (to compute candidates **p**, **q**)
  - **primality testing** (to check that candidates **p**, **q** are prime)
  - computation of the **GCD** (to verify that **e** and $\phi(n)$ are relatively prime)
  - computation of the **multiplicative inverse** (to compute **d** from **e**)

Modular powers

Repeated squaring algorithm

- speeds up computation of $\mathbf{a^p}$ mod **n**
- write the exponent **p** in binary
  - $\mathbf{p} = \mathbf{p_{b-1}}\,\mathbf{p_{b-2}} \cdots \mathbf{p_1}\,\mathbf{p_0}$
- start with $\mathbf{Q_1} = \mathbf{a^{p_{b-1}}}$ mod **n**
- repeatedly compute
  $\mathbf{Q_i} = ((\mathbf{Q_{i-1}})^2$ mod **n**$)\mathbf{a^{p_{b-i}}}$ mod **n**
- obtain $\mathbf{Q_b} = \mathbf{a^p}$ mod **n**

In total **O** (log **p**) arithmetic operations

Example

- 3^{18} mod 19 (18 = 10010)
- $\mathbf{Q_1} = 3^1$ mod 19 = 3
- $\mathbf{Q_2} = (3^2$ mod 19$)3^0$ mod 19 = 9
- $\mathbf{Q_3} = (9^2$ mod 19$)3^0$ mod 19 = 81 mod 19 = 5
- $\mathbf{Q_4} = (5^2$ mod 19$)3^1$ mod 19 =
  (25 mod 19)3 mod 19 = 18 mod 19 = 18
- $\mathbf{Q_5} = (18^2$ mod 19$)3^0$ mod 19 = (324 mod 19) mod 19 = 17·19 + 1 mod 19 = 1

Pseudo-primality testing

Testing whether a number is prime (**primality testing**) is a difficult problem

An integer **n** ≥ 2 is said to be a base-**x** **pseudo-prime** if

- $\mathbf{x}^{\mathbf{n}-1} \bmod \mathbf{n} = 1$ (Fermat's little theorem)

- Composite base-**x** pseudo-primes are rare
  - a random 100-bit integer is a composite base-2 pseudo-prime with probability less than 10^{-13}
  - the smallest composite base-2 pseudo-prime is 341
- Base-**x** pseudo-primality testing for an integer **n**
  - check whether $\mathbf{x}^{\mathbf{n}-1} \bmod \mathbf{n} = 1$
  - can be performed efficiently with the repeated squaring algorithm

Security properties

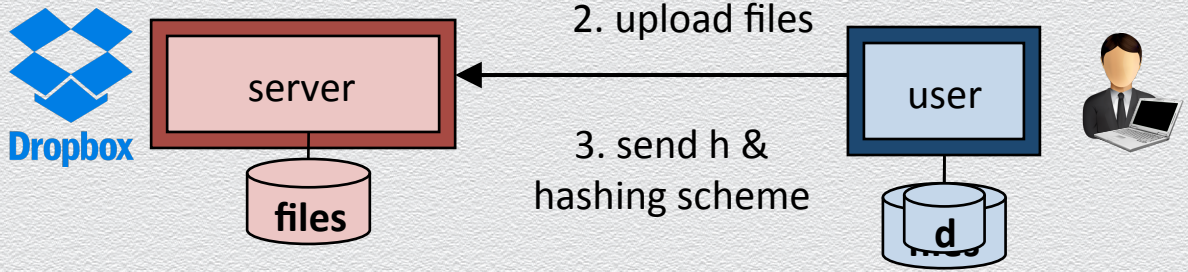
- Plain RSA is deterministic
  - why is this a problem?
- Plain RSA is also homomorphic
  - what does this mean?
  - multiply ciphertexts to get ciphertext of multiplication!
  - $[(m_1)^e \bmod N][(m_2)^e \bmod N] = (m_1 m_2)^e \bmod N$
  - however, not additively homomorphic

Real-world usage of RSA

- Randomized RSA
  - to encrypt message M under an RSA public key (e,n), generate a new random session AES key K, compute the ciphertext as $[K^e \bmod n, AES_K(M)]$
  - prevents an adversary distinguishing two encryptions of the same M since K is chosen at random every time encryption takes place
- Optimal Asymmetric Encryption Padding (OAEP)
  - roughly, to encrypt M, choose random r, encode M as $M' = [X = M \oplus H_1(r) , Y= r \oplus H_2(X)]$ where H_1 and H_2 are cryptographic hash functions, then encrypt it as $(M')^e \bmod n$

Database-as-a-service authentication model

Securing your cloud storage optimally – setup

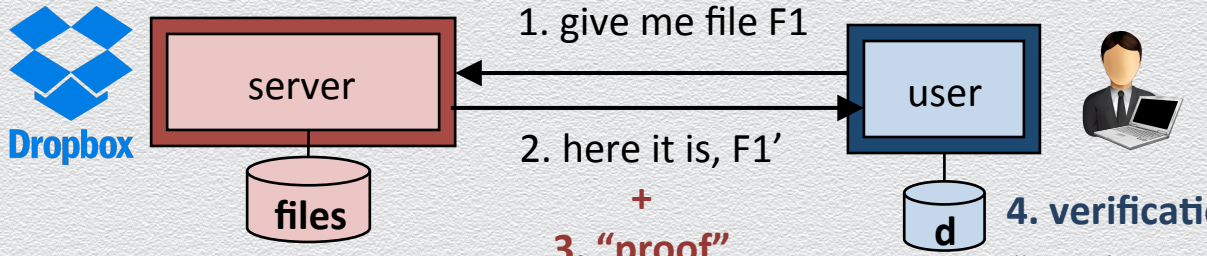


files = F = (F1, F2, …, F7, F8)

digest d is computed over all files

|d| << |F|

Securing your cloud storage optimally – data authentication



F = (F1, F2, …, F7, F8) **(or helper information)**

goals

security: Step 4 is reliable test

efficiency:

- d is as small as possible
- proof is as small as possible
- verification is as fast as possible

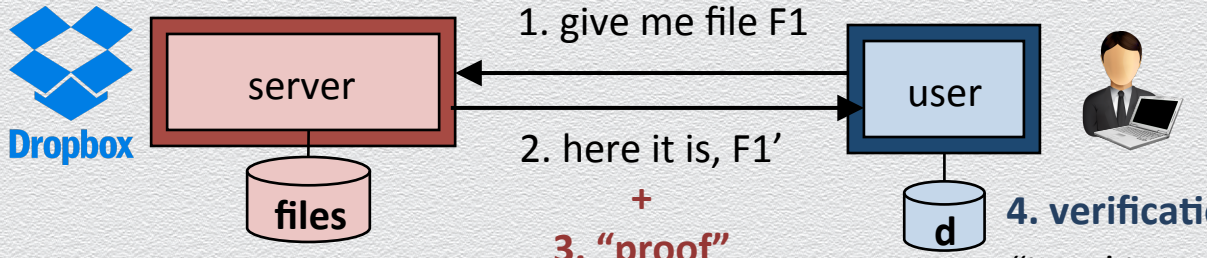
user has

- authentic digest d (locally stored)
- file F1' (to be checked)
- **proof** (to help checking integrity)

verification involves

- combine file F1' with the proof to re-compute candidate digest d'
- check if d' = d
- if yes, then F1 is intact; otherwise tampering is detected!

Hashing the files: Individually or as a whole?



$F = (F_1, F_2, \ldots, F_n)$

goals

security: Step 4 is reliable test

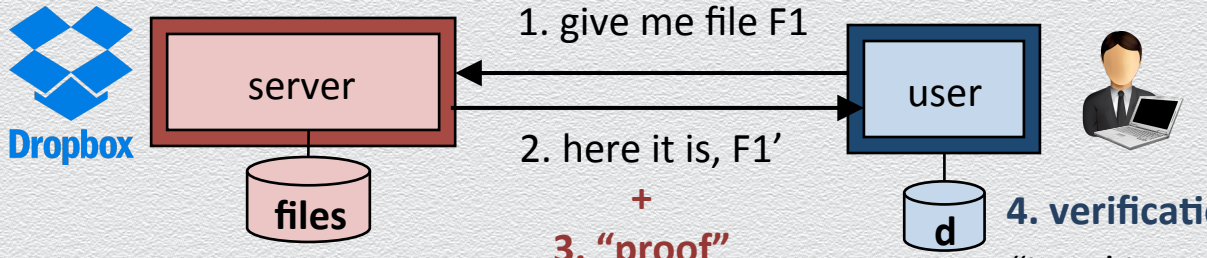
efficiency:

- d is as small as possible
- proof is as small as possible
- verification is as fast as possible

let $h_i = h(F_i)$, $1 \le i \le n$

$d = (h_1, h_2, \ldots, h_n)$ Vs. $d = h(h_1||h_2||\ldots||h_n)$ Vs. $d = h(F_1||F_2||\ldots||F_n)$

Hashing the files: In a chain or in a partition?



$F = (F_1, F_2, \ldots, F_n)$

let $h_i = h(F_i)$, $1 \le i \le n$

$d = h_n$, $h_i = h(F_i||h_{i-1})$, $1 < i \le n$, and $h_1 = F_1$

Vs.

k disjoint subsets each of n/k size

$d = h_n$, $h_i = h(\mathbf{h(F_i)}||h_{i-1})$, $1 < i \le n$, and $h_1 = \mathbf{h(F_1)}$

subsets & their digests are hashed in chains

goals

security: Step 4 is reliable test

efficiency:

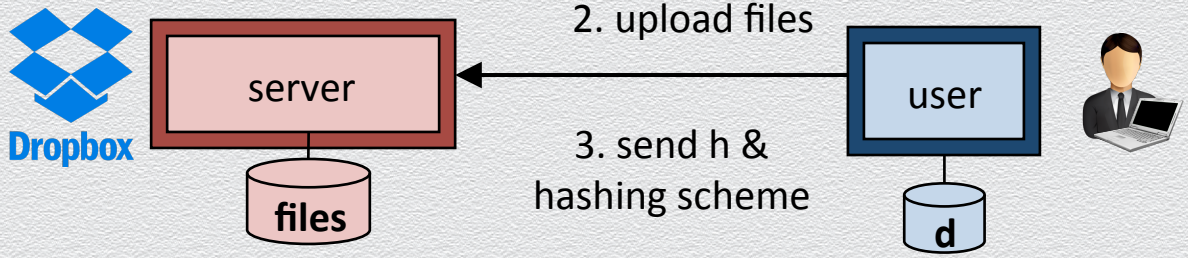
- d is as small as possible
- proof is as small as possible
- verification is as fast as possible

Towards an optimal hashing scheme

Lessons learned

- files should better be individually hashed as $h_i = h(F_i)$, $1 \leq i \leq n$, over which the final digest should be computed
- for $k > 2$:
  - hashing k files in a hash chain is preferable to hashing files as a whole
  - hash chains over k objects (hash values/files) result in **unbalanced** verification costs
    - e.g., proof size/verification time are sensitive to a file's position in the hash chain
- hashing file subsets individually across a balanced partition of the files:
  - offers trade-offs between space and verification costs
  - allows for hierarchical hashing schemes

Hashing via the Merkle tree



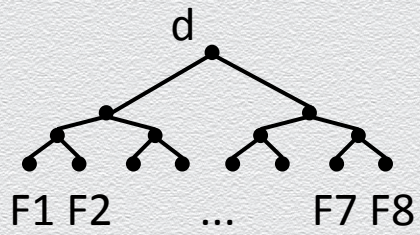
files = F = (F1, F2, …, F7, F8)

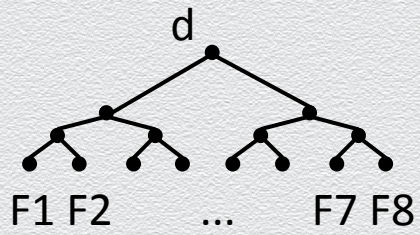
digest d is computed over all files

|d| << |F|

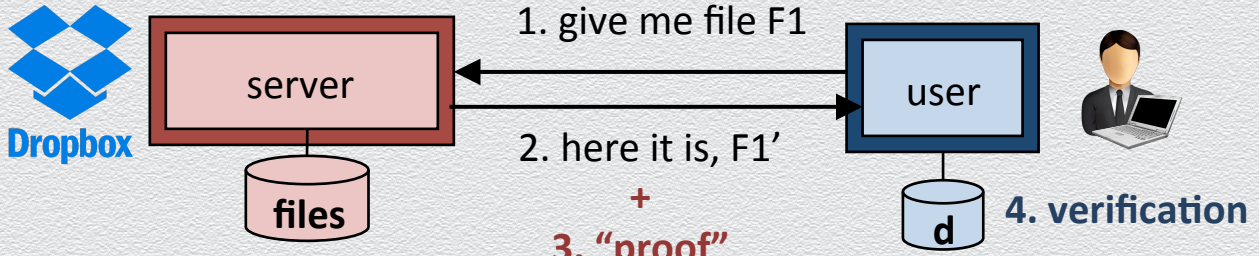
1. pre-process files
using CR hash function h

h: collision resistant iterative
hash function (e.g., SHA-2)





The Merkle tree



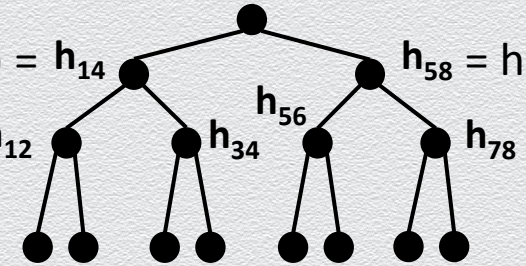
$F = (F_1, F_2, \ldots, F_8)$

$d = h (h_{14} \,||\, h_{48})$

$h (h_{12} \,||\, h_{34}) = h_{14}$

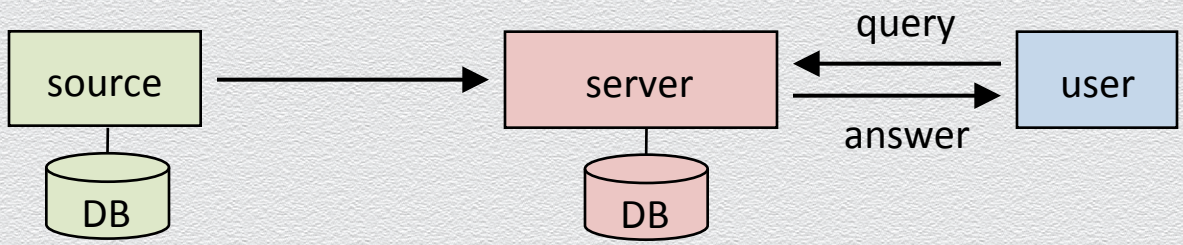
$h_{58} = h (h_{56} \,||\, h_{78})$

$h (h_1 \,||\, h_2) = h_{12}$



let $h_i = h(F_i)$, $1 \le i \le n$

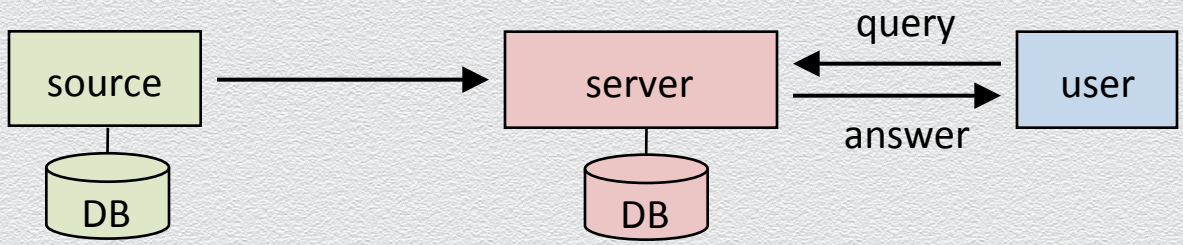
Generalization: DB-as-a-service authentication model



cloud-based DB

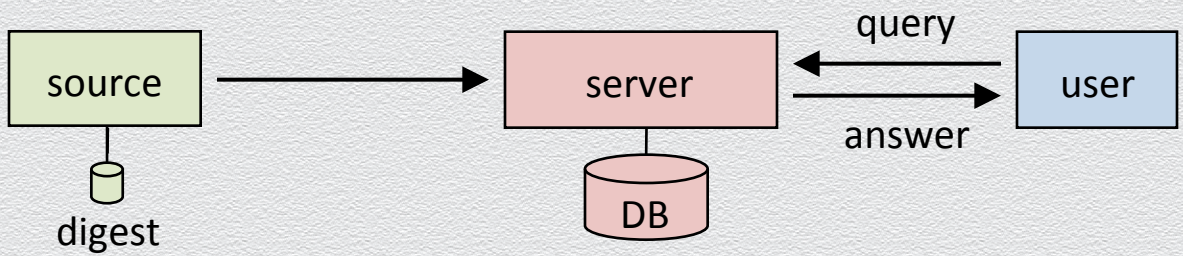
portals, hubs

Generalization: DB-as-a-service authentication model



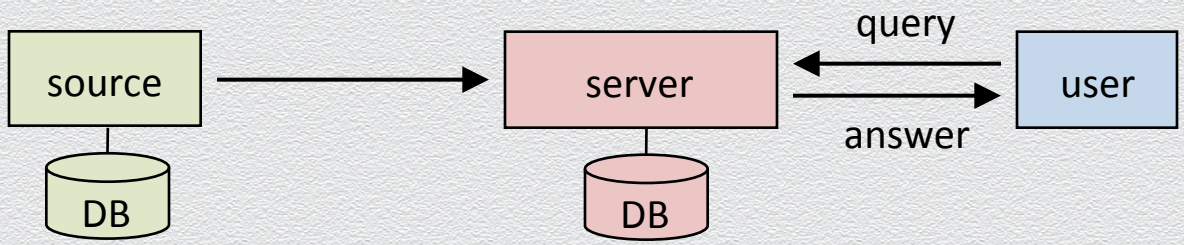
Internet protocols
(DNS, OCSP)

Generalization: DB-as-a-service authentication model



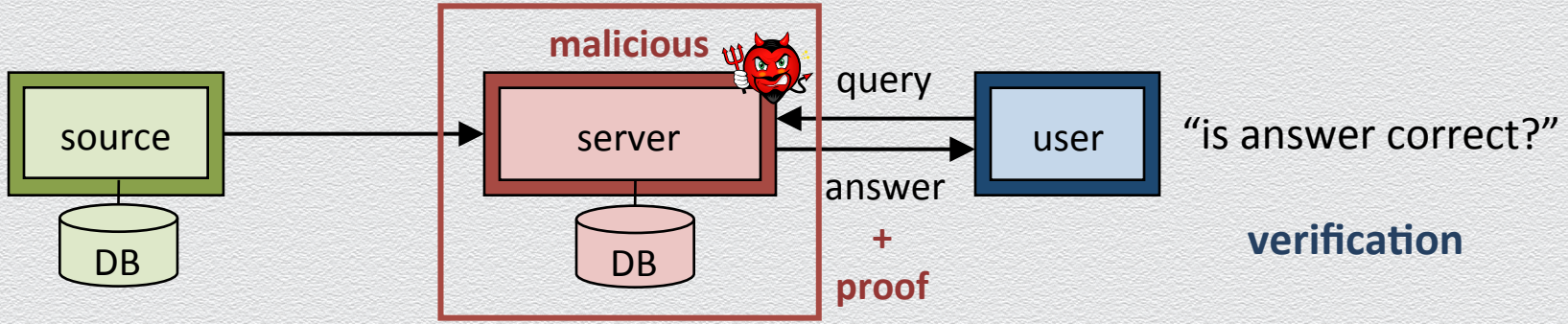
file hosting

DB-as-a-service authentication model



Integrity checks offer authenticated queries ⇒ guarantees correct answer, as if coming directly from the source!

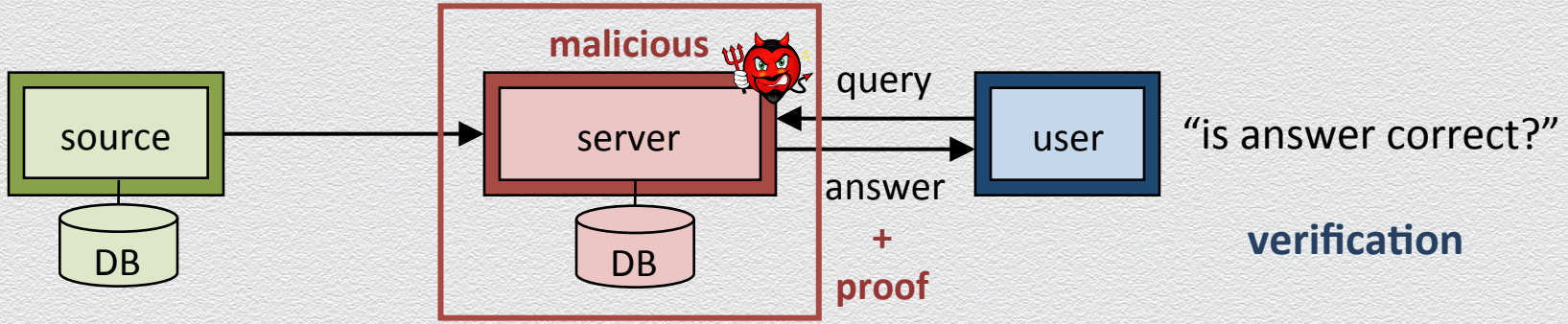
DB-as-a-service authentication model



Integrity checks offer authenticated queries

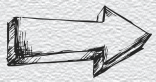
- crypto-based: harden data/computations to provide verifiable answers
- reliable: allow no false positives or negatives ⇒ crypto does its work!

DB-as-a-service authentication model

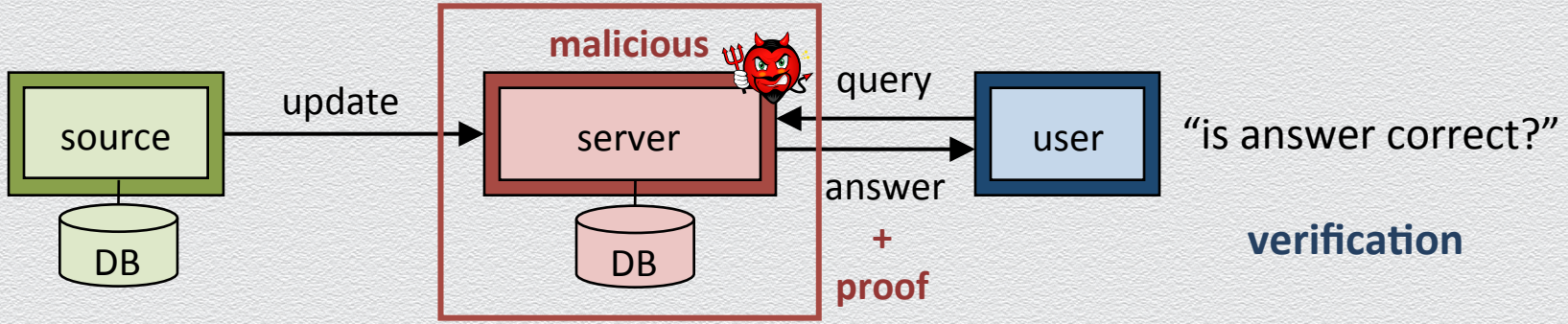


Integrity checks offer authenticated queries

- crypto-based: harden data/computations to provide verifiable answers
- reliable: allow no false positives or negatives
- utility-preserving: practical & easy to adopt
  - fast times for query, verification
  - near total storage, low bandwidth usage for proof

 minimize all overheads

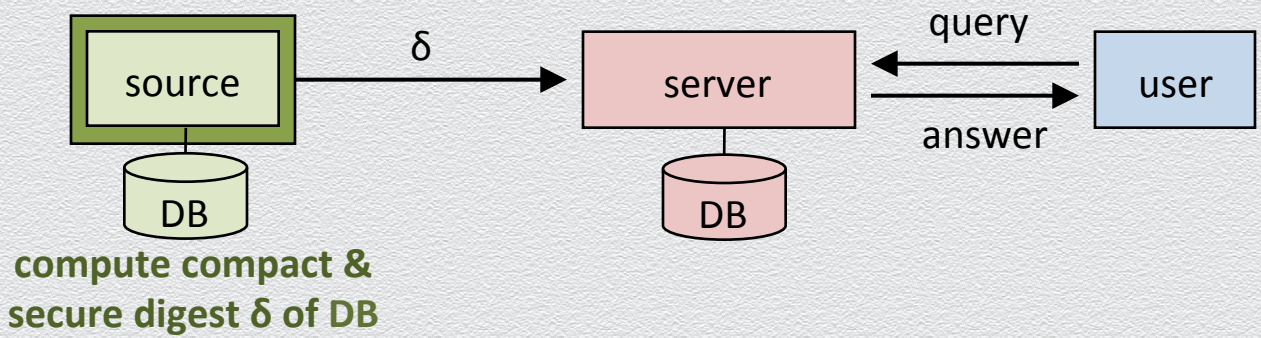
DB-as-a-service authentication model



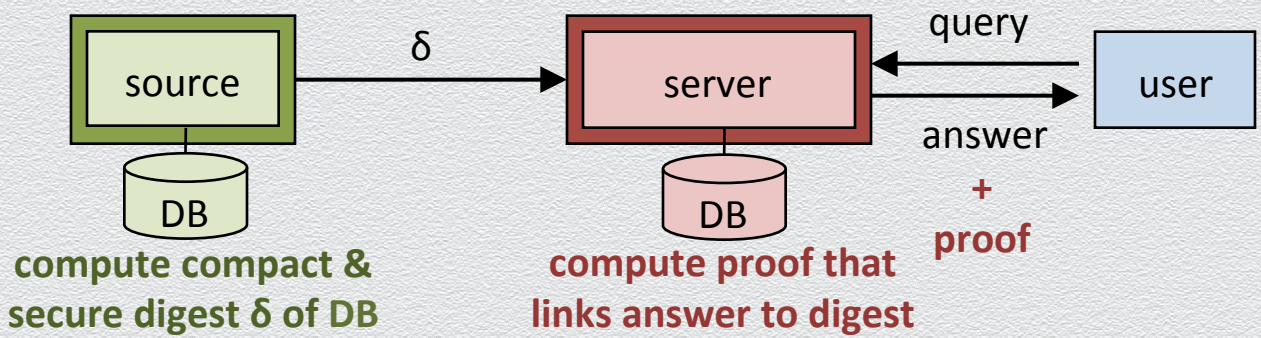
Integrity checks offer authenticated queries

- crypto-based: harden data/computations to provide verifiable answers
- reliable: allow no false positives or negatives
- utility-preserving: practical & easy to adopt
  - fast times for query, verification, update
  - near total storage, low bandwidth usage for proof, update

Intro to authenticated querying (AQ101)

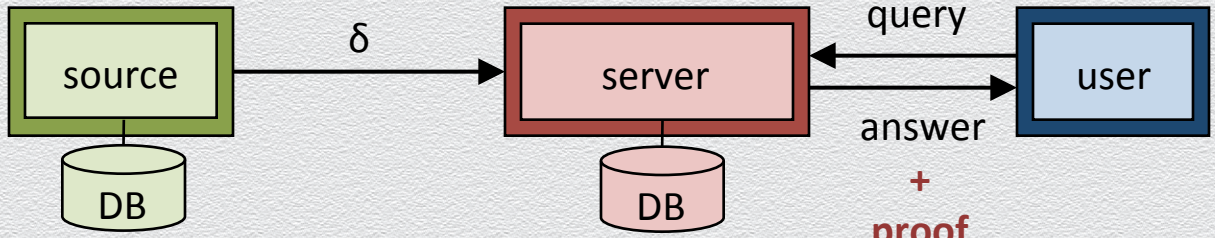


Intro to authenticated querying (AQ101)



Intro to authenticated querying (AQ101)

Assumption: user possesses authentic δ



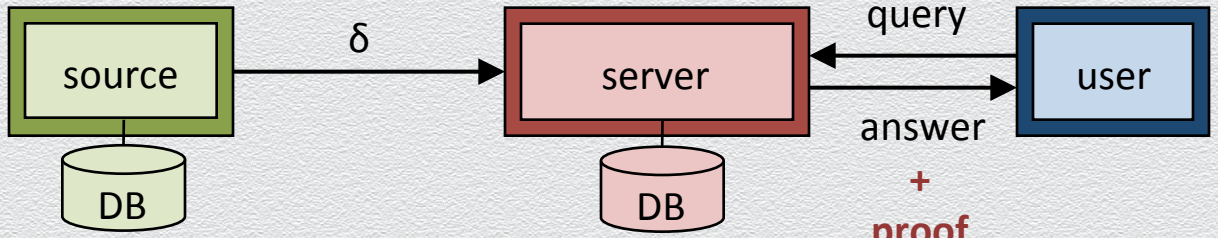
compute compact & secure digest δ of DB

compute proof that links answer to digest

verification

use proof to securely link answer to authentic digest

Intro to authenticated querying (AQ101)



compute compact & secure digest δ of DB

compute proof that links answer to digest

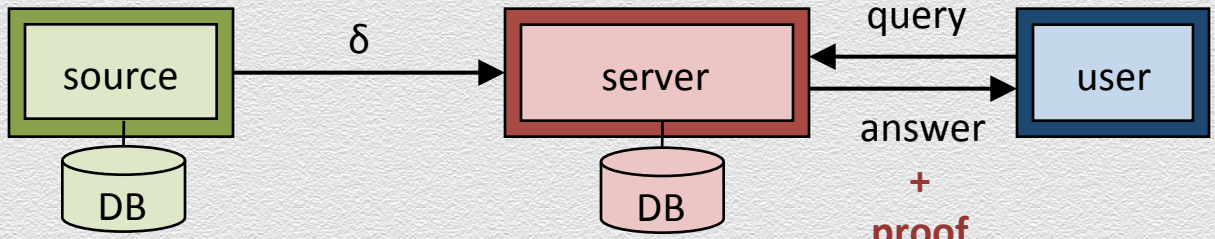
Assumption: user possesses authentic δ

easily removed using PKI, digital signatures

verification

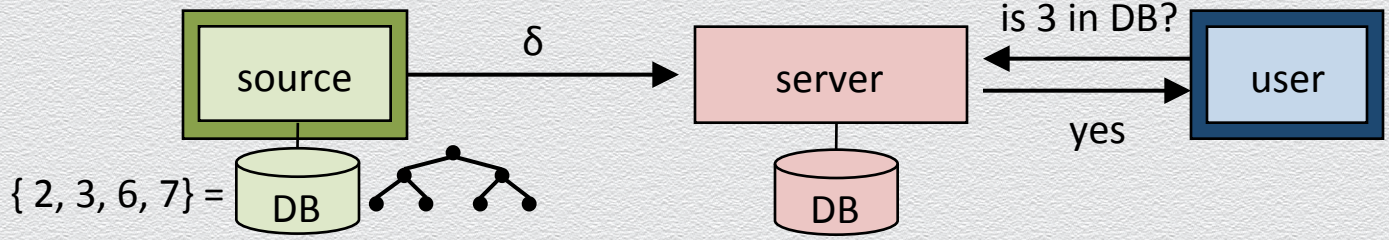
use proof to securely link answer to authentic digest

Intro to authenticated querying (AQ101)



Using digital signatures to provide authentic digest δ

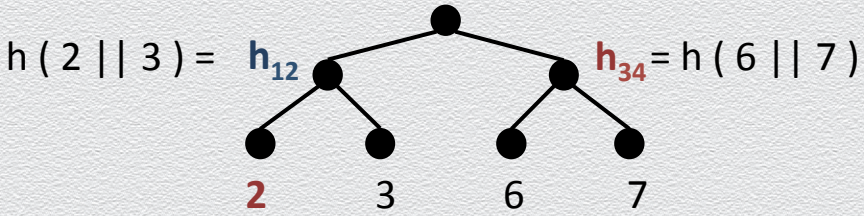
Example: Set membership – digest computation



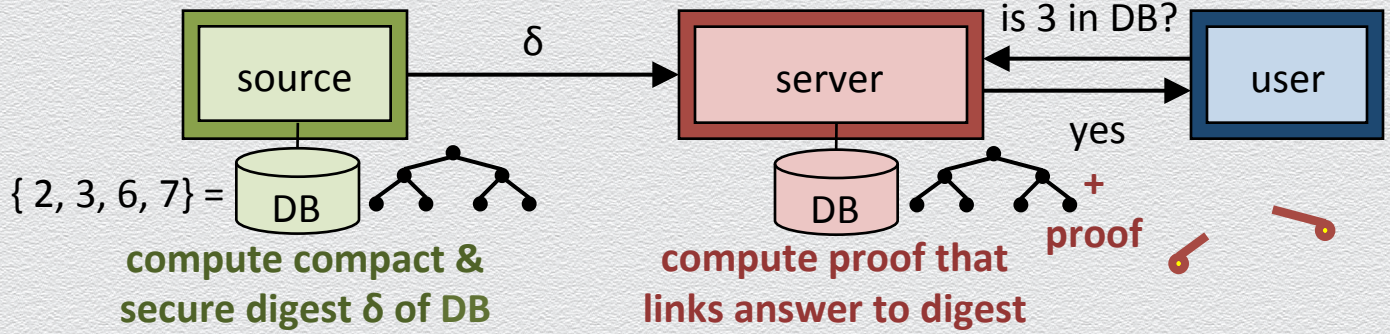
compute compact & secure digest δ of DB

Merkle tree hash $\delta = h (h_{12} || h_{34})$

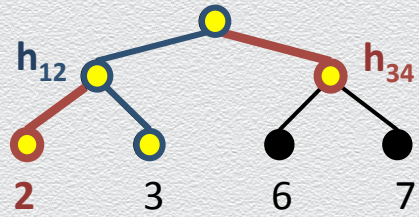
h: collision resistant terative hash function (e.g., SHA-2)



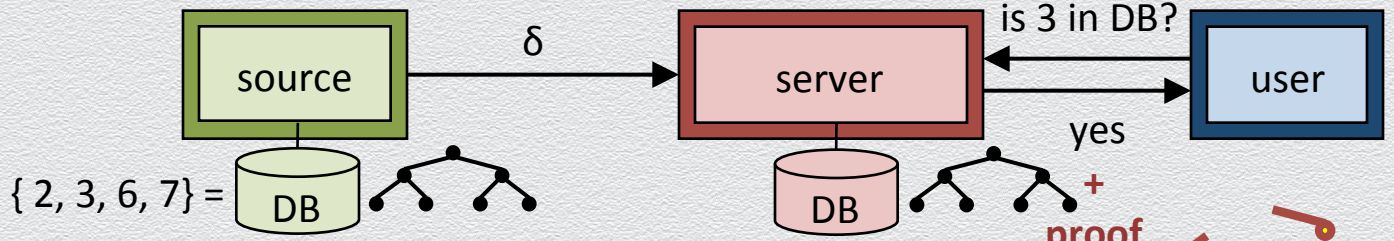
Example: Set membership – proof computation



Merkle tree hash δ



Example: Set membership – proof verification



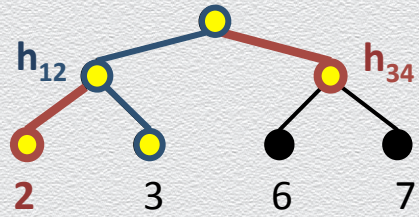
verification
(given δ)

compute compact & secure digest δ of DB

compute proof that links answer to digest

use proof to securely link answer to authentic digest

Merkle tree hash δ



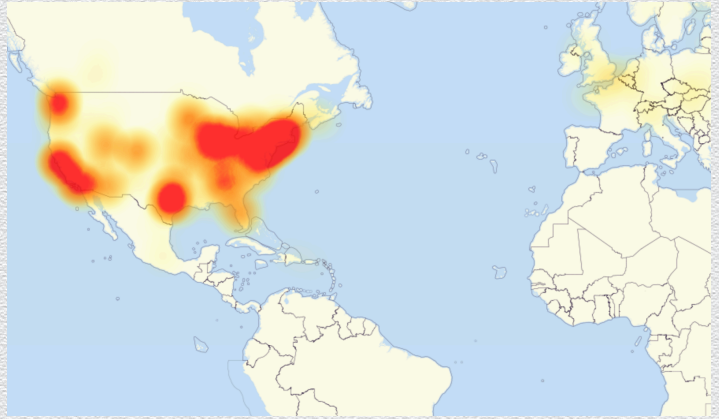
i.e.:
recompute h12
recompute δ
compare against given δ

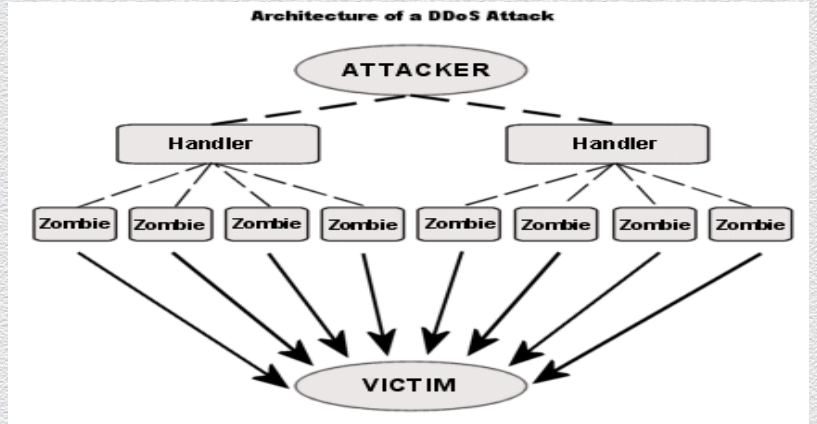
Dyn DDoS attack

It's unfair! – I had no class but couldn't watch my Netflix series!

On October 21, 2016, a large-scale cyber was launched

- it affected globally the entire Internet but particularly hit U.S. east coast
- during most of the day, no one could access a long list of major Internet platforms and services, e.g., Netflix, CNN, Airbnb, PayPal, Zillow, …
- this was a **Distributed Denial-of-Service (DDoS)** attack





DoS: A threat (mainly) against availability

Which main security property does a Denial-of-Service (DoS) attack attempt to defeat?

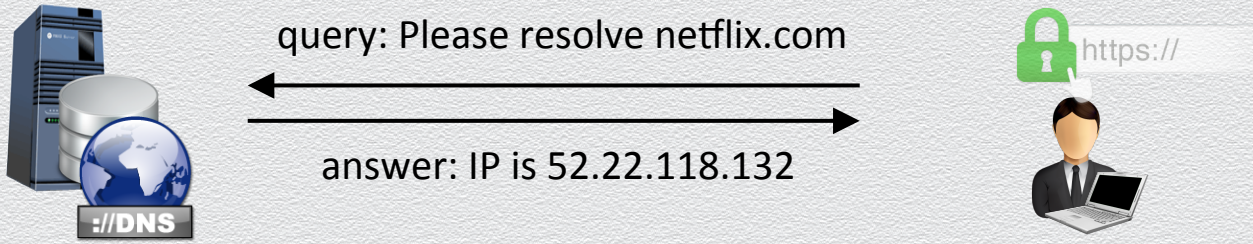
- availability; a user is denied access to authorized services or data
  - availability is concerned with preserving authorized access to assets
  - a DoS attack aims against this property; its name itself implies its main goal
- integrity & confidentiality; services or data are modified or accessed by an unauthorized user
  - elements of a DoS attack may include breaching the integrity or confidentiality of a system
  - but the end goal is disruption of a service or data flow; not the manipulation, fabrication or interception of data and services

DNS security

The Domain Name Service (DNS) protocol

Resolving domain names to IP addresses

- when you type a URL in your Web browser, its IP address must be found
  - e.g., domain name “netflix.com” has IP address “52.22.118.132”
  - larger websites have multiple IP responses for redundancy to distributing load
- at the heart of Internet addressing is a protocol called DNS
  - a database translating Internet names to addresses

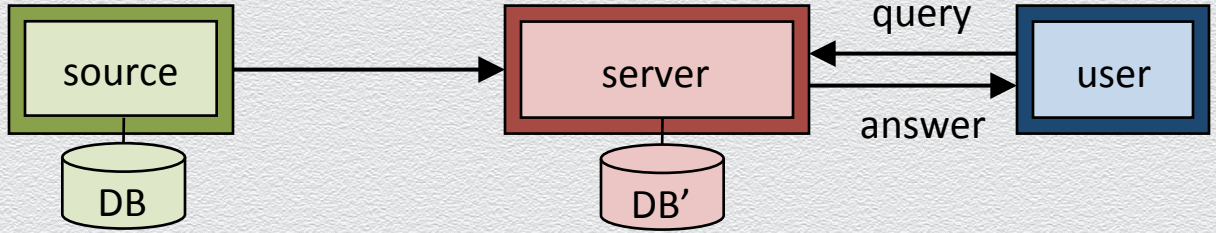


DNS name resolution is a critical asset – a target itself!

What main security properties must be preserved in such an important service?

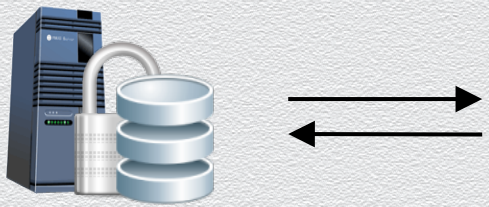
- all properties in CIA triad are relevant!
- resolving domain names to IP addresses is a service that
  - must critically be available during all times – availability
    - or else your browser does not know how to connect to Netflix…
  - must critically be trustworthy – integrity
    - or else connections to malicious sites may occur (e.g., DNS-spoofing attacks)
  - must also protect database entries that are not queried – confidentiality
    - or else an attacker may find out about the structure of a target organization (e.g., zone-enumeration attacks)

DNS as a (distributed) database-as-a-service



DNS entries:
<netflix.com, 52.22.118.132>

subset of cached queried entries
(or information of other resolvers)



"primary"
name server

"secondary"
name server

please resolve netflix.com

IP is 52.22.118.132
(or "aWa2j3netflix.com
is a non-existent domain")

Recursive name resolution: hierarchical search

Search is performed recursively and hierarchically across different type of DNS resolvers

- application-level (e.g., Web browser), OS-level (e.g., stub resolver): locally managed
- recursive DNS servers: query other resolvers and cache recent results

DNS entries:
<netflix.com, 52.22.118.132>

primary

subset of cached queried entries
(or information of other resolvers)

secondary

netflix.com

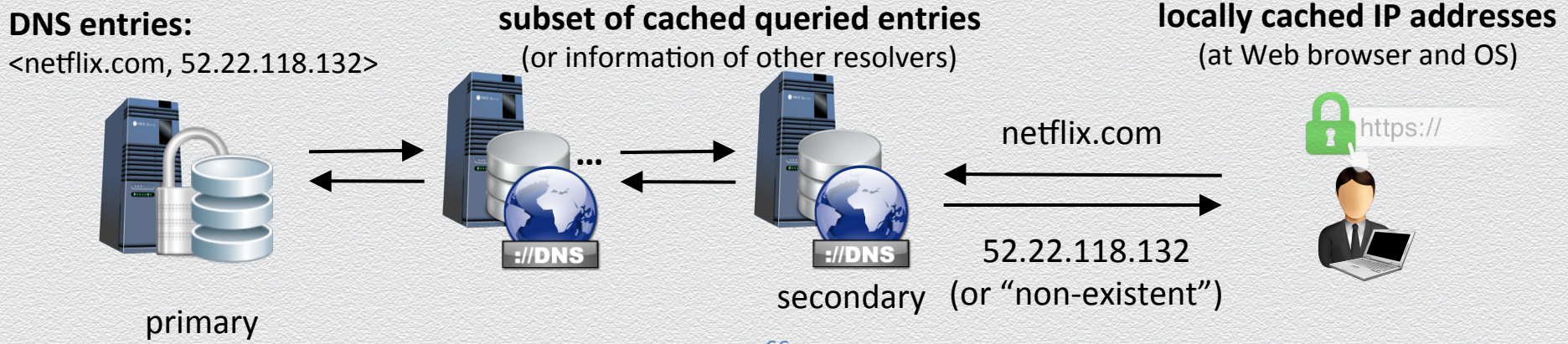
52.22.118.132
(or "non-existent")

locally cached IP addresses
(at Web browser and OS)

Recursive name resolution: hierarchical search

Search is performed recursively and hierarchically across different type of DNS resolvers

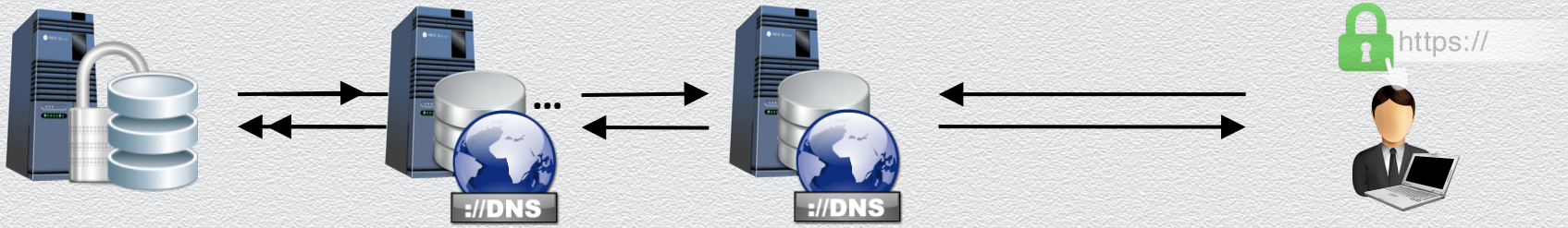
- application-level (e.g., Web browser), OS-level (e.g., stub resolver): locally managed
- recursive DNS servers: query other resolvers and cache recent results
- root name servers: refer to appropriate TLD (top-level domain) server
- TLD servers: control TLD zones such as .com, .org, .net, etc.



Recursive name resolution: flexibility

Infrastructure allows for different configurations

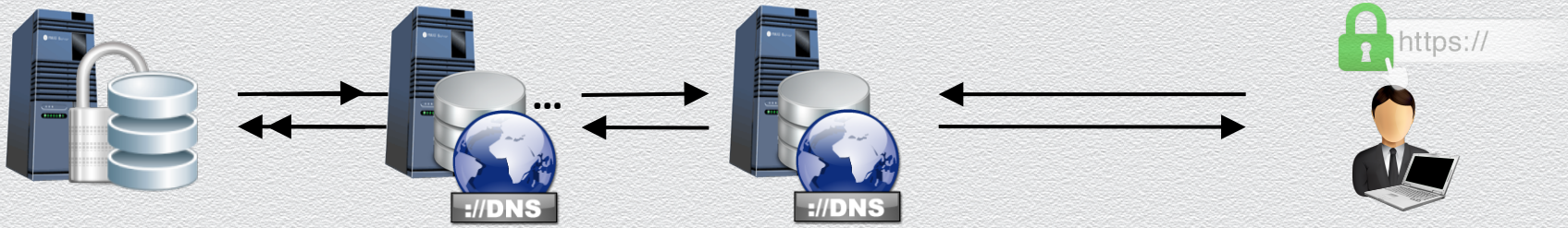
- authoritative-only servers: answer queries on zones they are responsible for
  - fast resolution, no forwarding, no cache
- caching / forwarding DNS servers: answer queries on any public domain name
  - recursive search / request forwarding, caching for speed, first-hop resolvers
- master / slaves DNS servers: authoritative servers replicating DNS data of their domains
- public / private DNS servers: control access to protected resources within an organization



Recursive name resolution: benefits

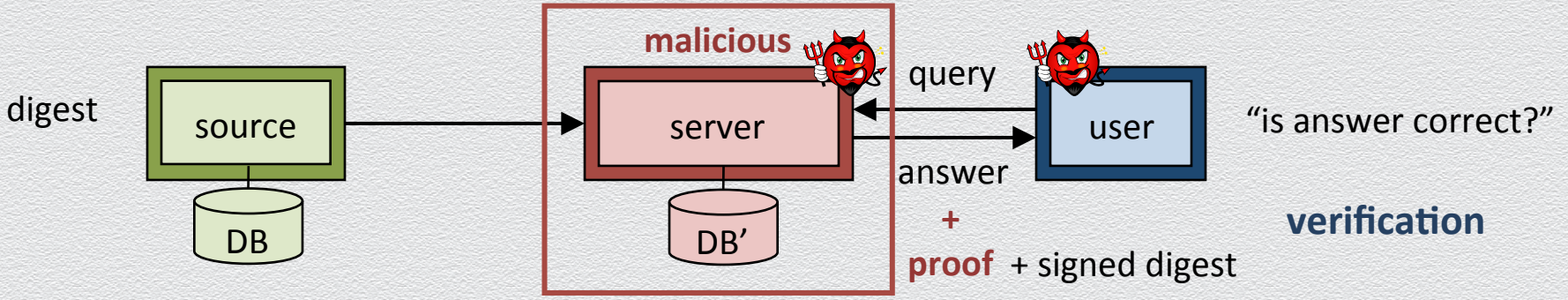
Why DNS uses non-authoritative name servers (that is, recursive resolution)?

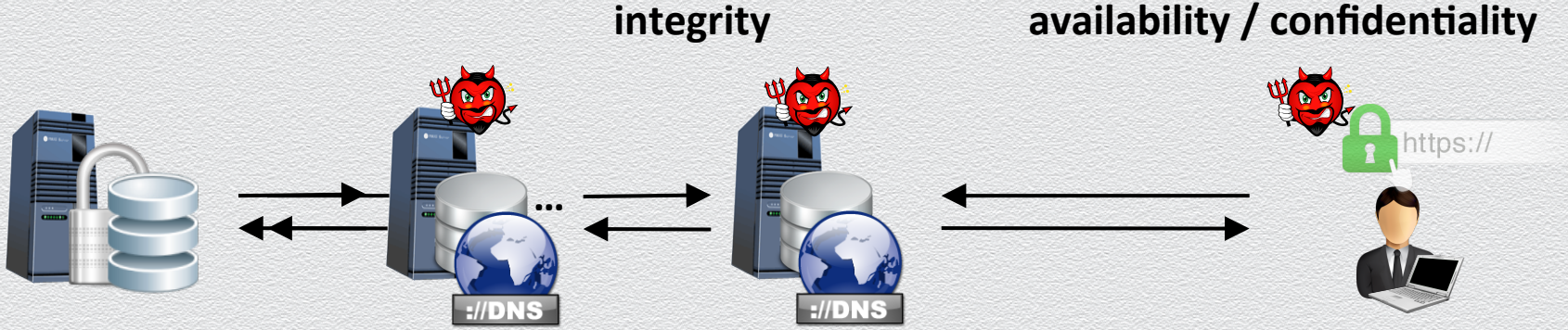
- for more scalability & locality
  - high query loads can saturate the response capacity of primary servers
  - secondary do not have to store large volumes of DNS entries
  - cached recently queried domain names speed up searches due to locality of queries
- for added security / locality / scalability alone – not quite
  - e.g., non-authoritative name servers are untrusted and thus possibly compromised



DNSSEC & NSEC3

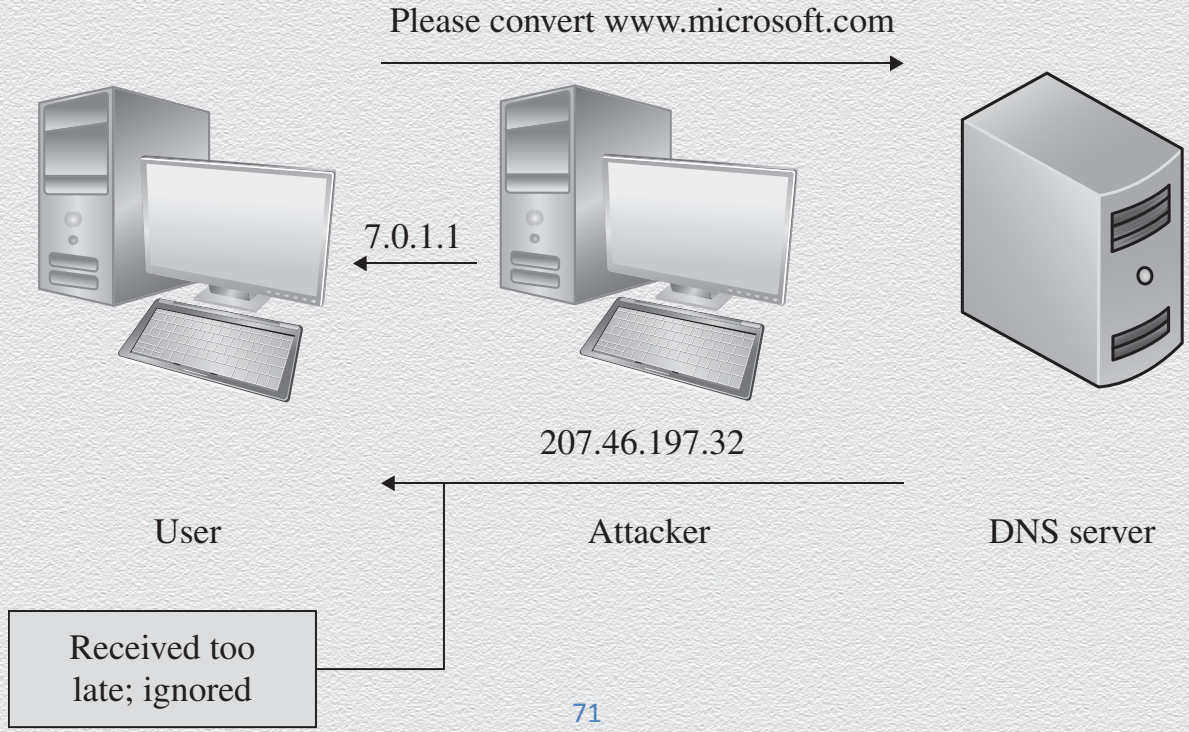
A critical asset prone to attacks





DNS spoofing (or cache poisoning)

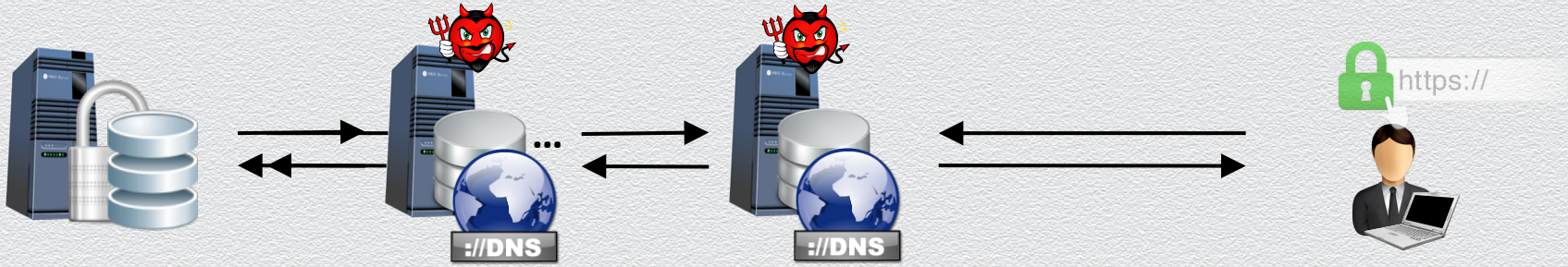
The attacker acts as the DNS server in order to redirect the user to malicious sites



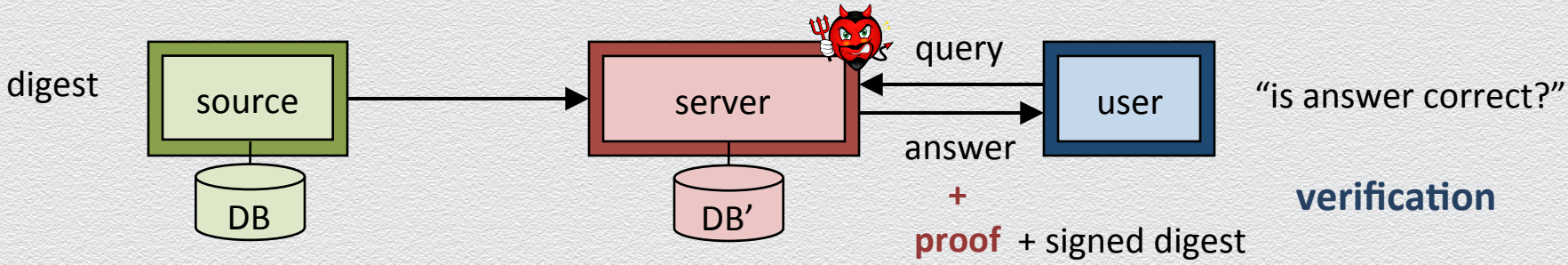
DNSSEC (& NSEC)

Security extension of DNS protocol to protect integrity of DNS data

- correct resolution, origin authentication, authenticated denial of existence
- specifications made by Internet Engineering Task Force (IETF) via RFCs
  - an RFC (request for comments) is a suggested solution under peer review
- challenges: backward-compatible, simplicity, confidentiality, who signs
  - NSEC (next secure record): extension that provides proofs of denial of existence



DNSSEC & NSEC: core idea

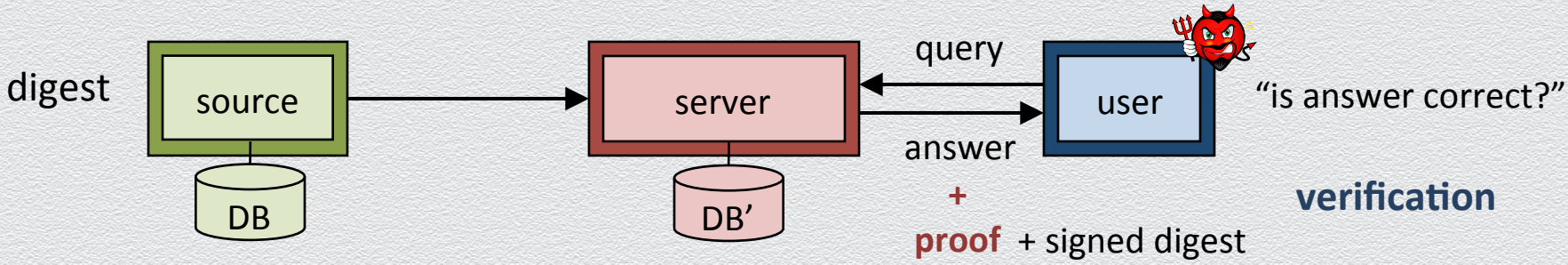


DNSSEC protocol: each DNS entry is pre-signed by primary name server

NSEC protocol:

- domain names are lexicographically ordered and then each pair of neighboring existing domain names is pre-signed by the primary name server
- non-existing names, e.g., aWa2j3netflix.com are proved by providing this pair "containing" missed query name, e.g., <awa.com, awb.com>

From NSEC to NSEC3



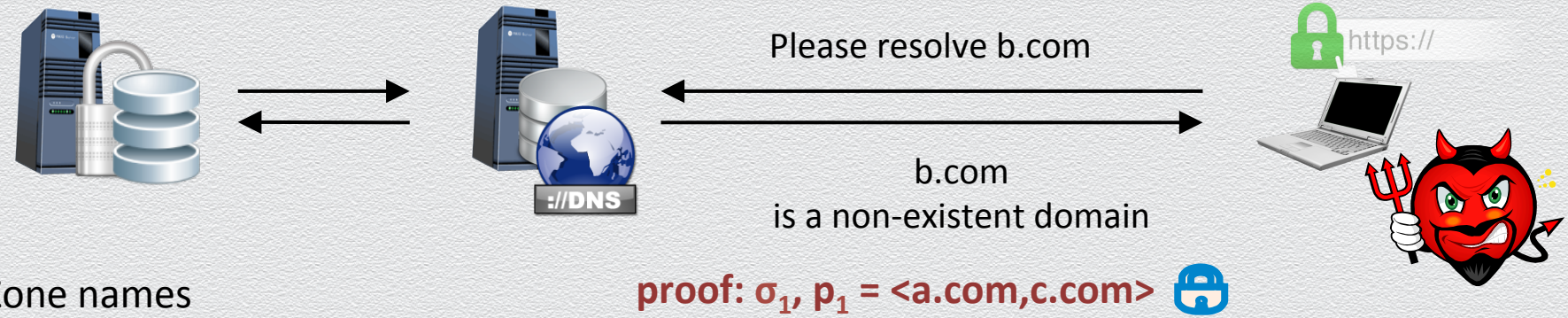
Vulnerability in NSEC protocol

confidentiality

- proofs of non-existing names reveal information about domain names
- an attacker can simply as a "querier" learn information about the network structure of a target organization!

DNS zone enumeration attack

The attacker acts as the user in order to learn the domain names of an entire zone



Zone names

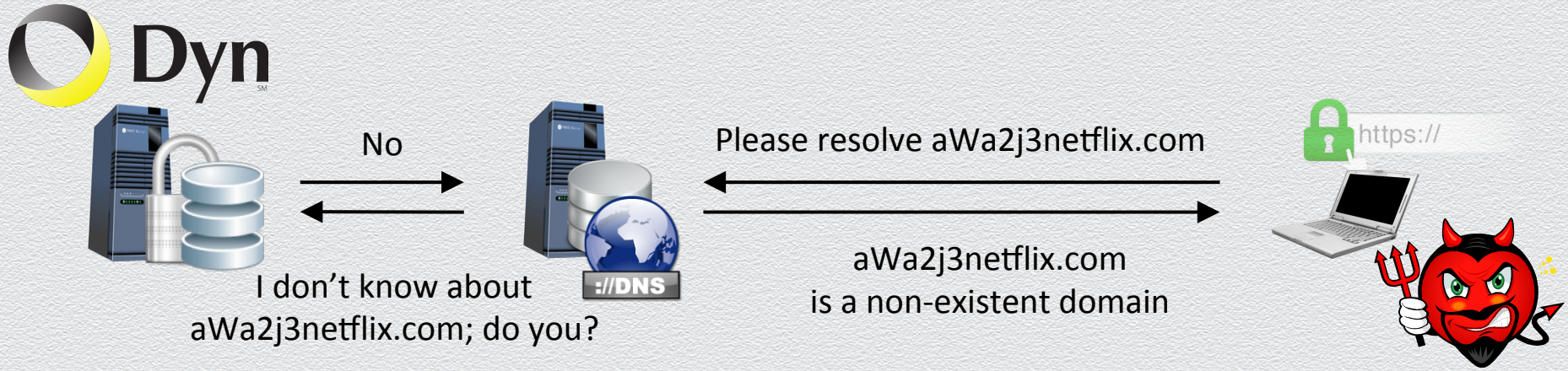
- a.com
- c.com
- e.com
- z.com

proof: σ_1, p_1 = <a.com,c.com>

confidentiality

DDoS attacks

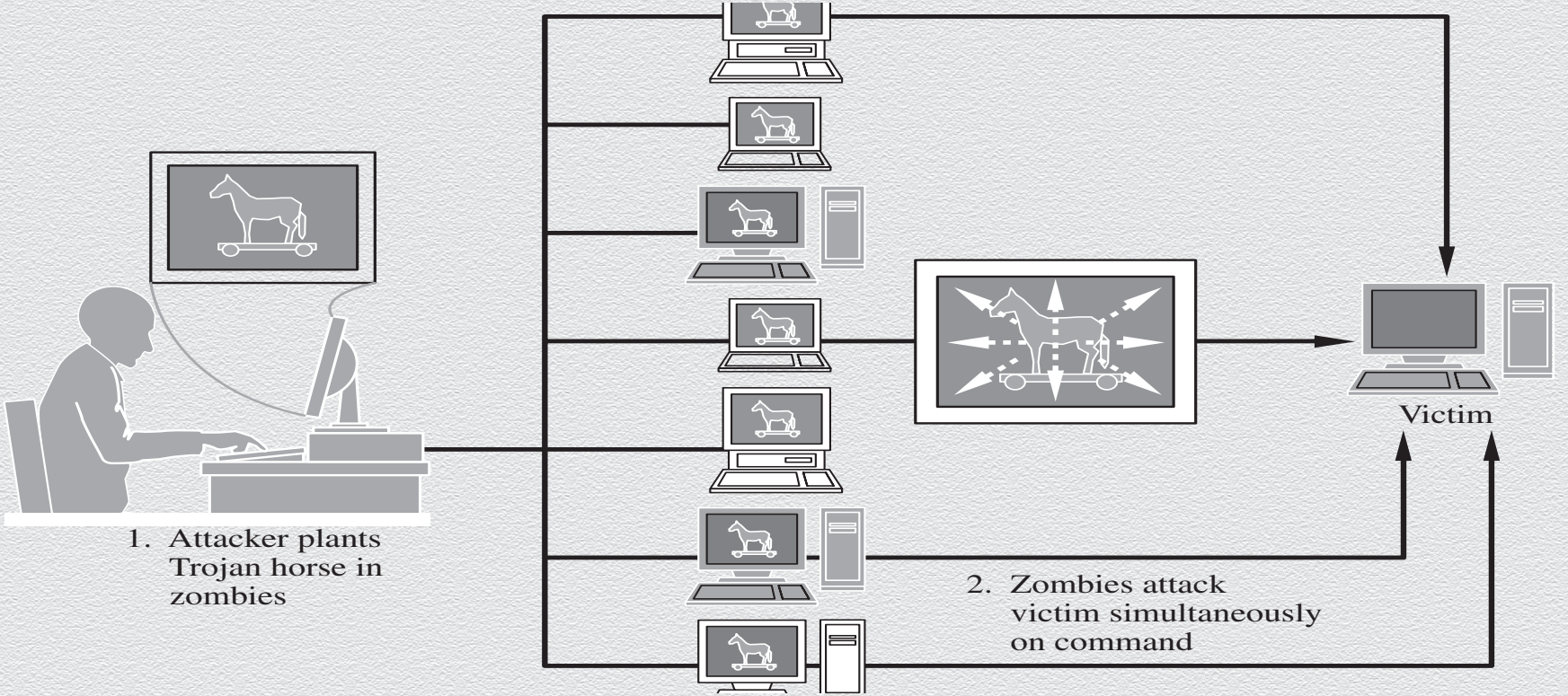
Core idea of attack: Saturate Dyn's primary servers



Attack:

- from a compromised machine ask for domain names that do not exist
- query is forwarded to fewer primary Dyn servers, i.e., defeating benefits of distribution
- ask **A LOT** of such queries to bring down the Dyn DNS service!

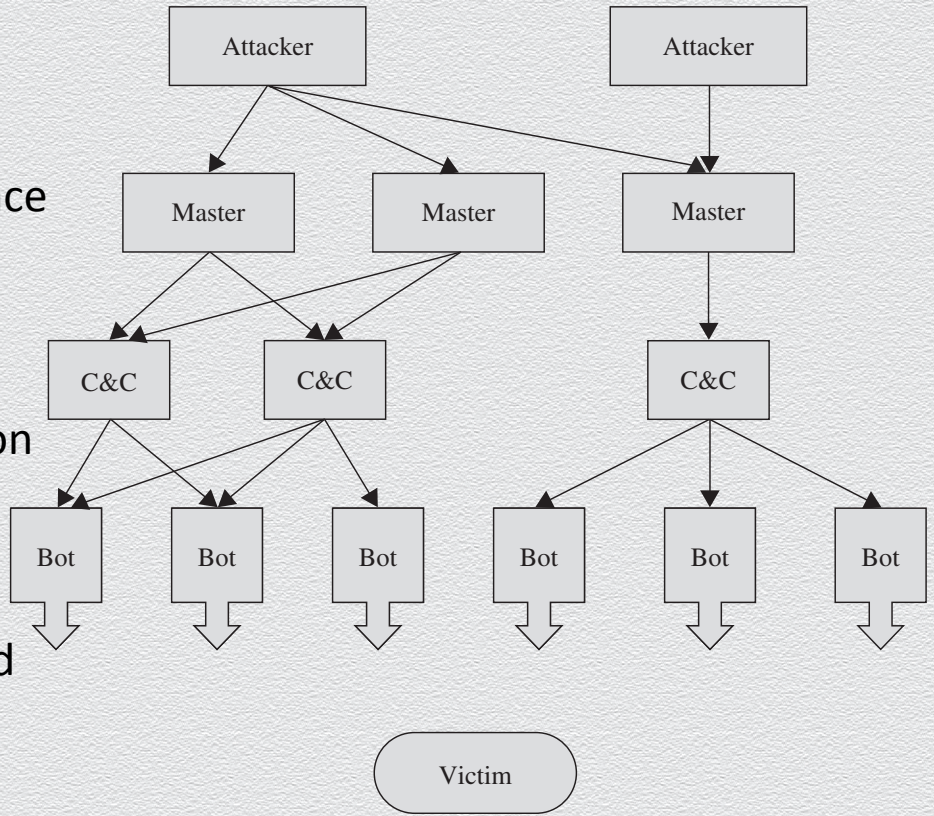
Distributed Denial of Service (DDoS)



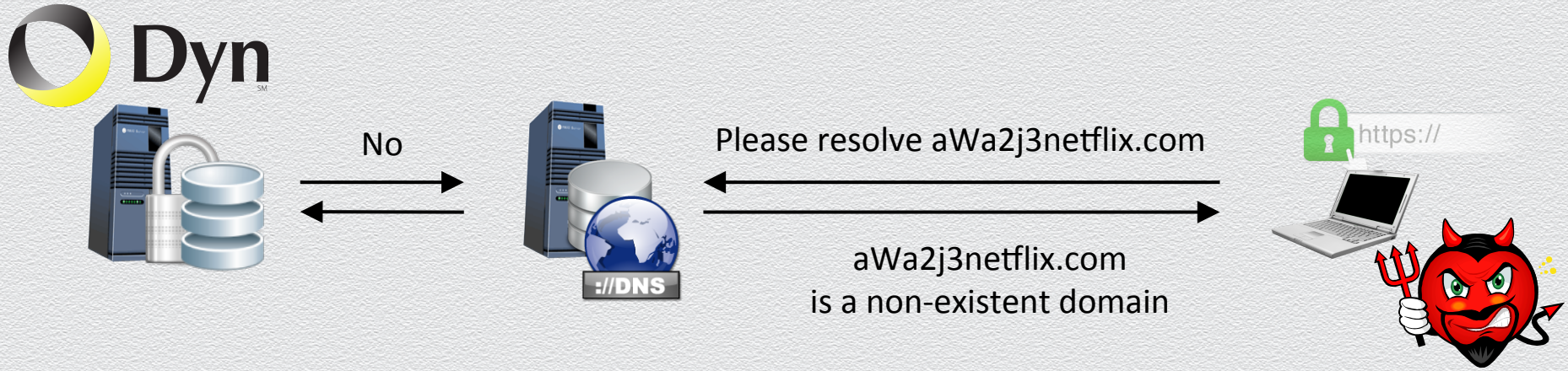
Botnets

Networks of machines running malicious code under remote control

- massive: scale to million of bots
  - comprise main tool for DDoS attacks
- stealth: remain undetected & difficult to trace
  - do little harm to the host machines
    - users won't likely remove malware
  - multiple-level attacker Vs. bots separation
- resilient: have redundant components
  - even if one master or C&C node is taken down, connectivity is maintained

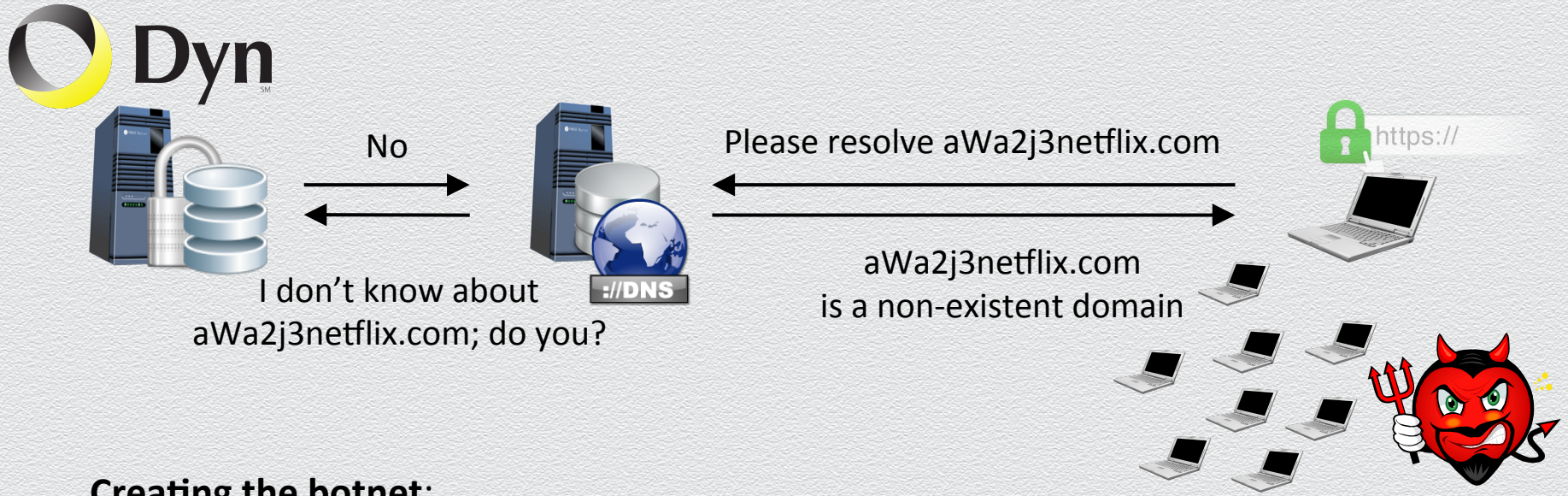


Why botnets are often behind DoS attacks?



- to avoid effective countermeasures and increase "attack" traffic
  - if the high-volume "attack" traffic comes from few devices, they can be filtered out by blocking their connections to the Dyn servers
  - by employing a large botnet of millions of devices the attacker inflicts a larger, more devastating "attack" traffic against the victim Dyn servers

Recruiting an army: Internet of Things (IoT)



Creating the botnet:

- compromise easy targets: IoT "thin" devices, e.g., printers, cameras, home routers, …
- how? find a vulnerability on these devices...
  - all such devices used an OS with a static, hard-wired, thus known, admin password...!

The Internet of Things (IoT)

Refers to Internet-connected everyday devices

- comprise a world of so-called smart devices
- examples:
  - smart appliances, such as refrigerators and dishwashers
  - smart home, such as thermostats and alarm systems
  - smart health, such as fitness monitors and insulin pumps
  - smart transportation, such as driverless cars
  - smart entertainment, such as video recorders
- potential downsides
  - loss of privacy
  - loss of control of data
  - potential for subversion
  - mistaken identification
  - uncontrolled access

Smartphones

The control hub of the IoT – important target for malware

- 2013: 143,211 distinct new forms of malware against mobile devices
- 98% targeted Android devices, far in excess of its market share
  - Android: open approach
    - unlike its competitors, does not limit the software users are allowed to install
    - thus, an easier target
  - Apple: locked-down approach
    - in contrast, only allows apps from its app store to be installed on its smartphones
    - all apps go through an approval process, which includes some security review
    - once approved, apps are signed, using a certificate approach

More generally on DoS attacks

DoS attacks are attempts to defeat a system's availability

- volumetric attacks (e.g., flooding)
  - potential weaknesses in the capacity of a computer network

insufficient capacity, overloading

- application-based attacks (e.g., routing malfunction, blocked access)
  - exhaust the resources of an application that services a particular network

blocked access

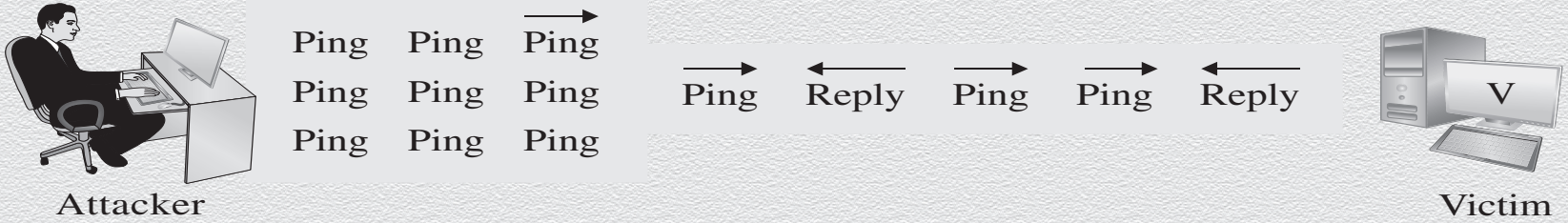
- disabled communications (e.g., access failure)
  - physically disable a communication link or disrupt a single-point of failure
- hardware or software failure (e.g., access failure)
  - failures on machines or programs (without fault-tolerance provisions)

unresponsive asset

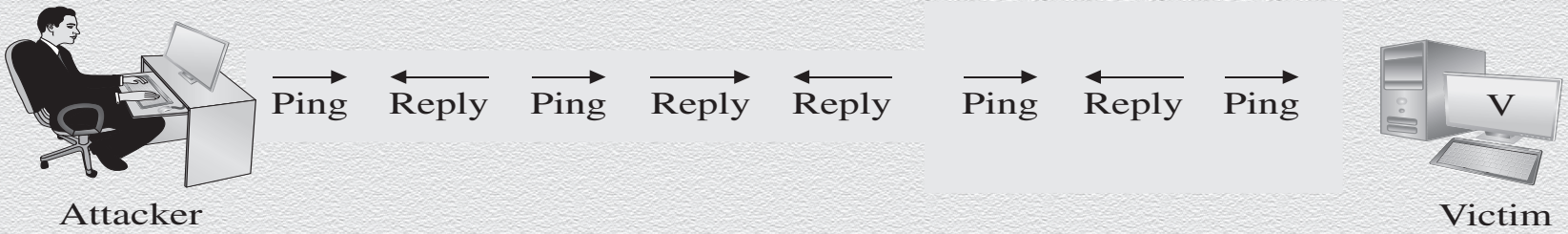
Examples

- Benign errors
  - Beth Israel Hospital system downtime, in 2002, due to mishandling of switches
- Malicious code
  - use vulnerabilities in communication protocols
  - e.g., cause uncontrolled congestion in TCP communications (Vs. UDP)
  - e.g., exploit/misuse Internet Control Message Protocols (ICMP)
    - ping (destination is reachable and functional)
    - echo (the connection between two machines is reliable)
    - destination unreachable (destination address cannot be accessed)
    - source quench (destination is saturated and source should suspend transmissions)

Ping flood (or pink of death)



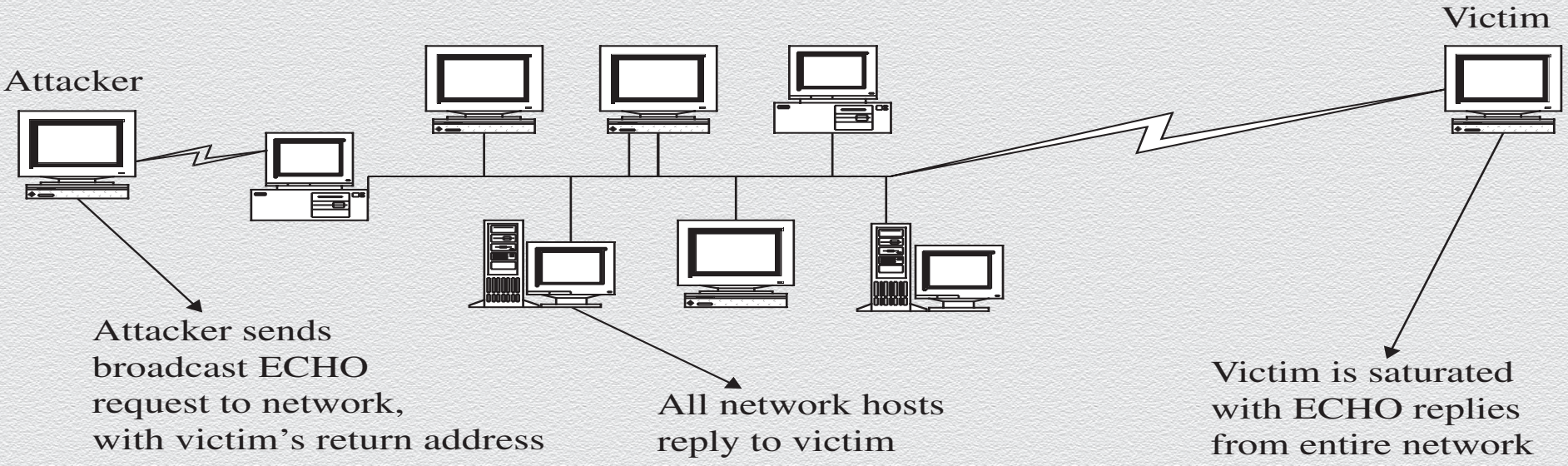
(a) Attacker has greater bandwidth



(b) Victim has greater bandwidth

Smurf Attack

Unwitting victims become accomplices



Echo-Chargen

Victim A

Victim B

Chargen packet with echo bit on →

← Echoing what you just sent me

Chargen another packet with echo bit on →

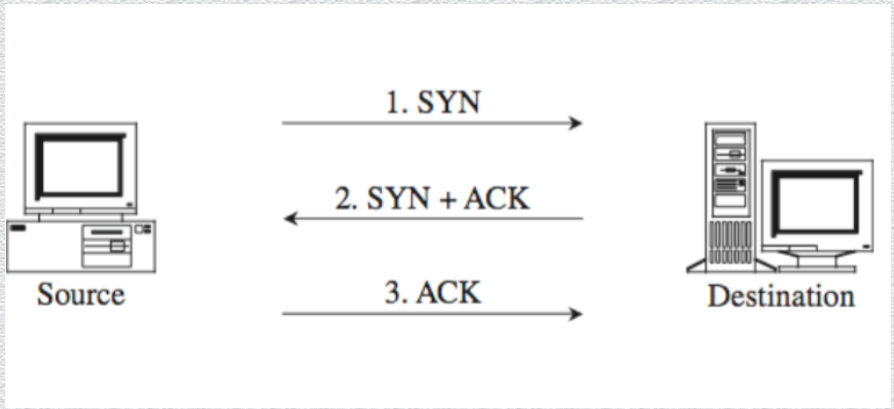
← Echoing that again

Chargen another packet with echo bit on →

SYN flood

Three-way handshake used in TCP to establish a new session

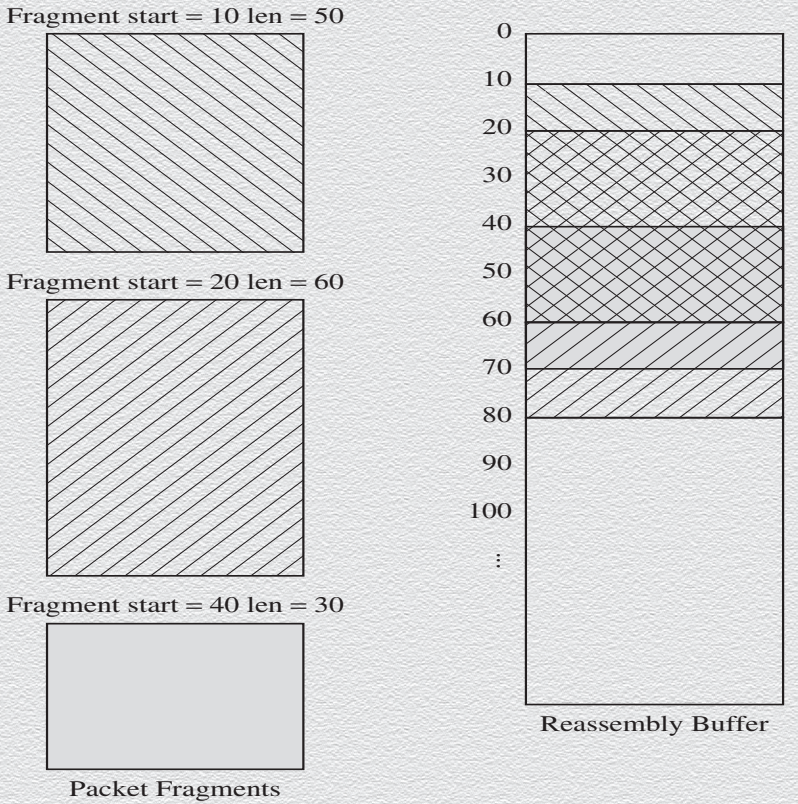
- source & destination exchange control messages to complete session creation
- destination keeps track of incomplete session-creation protocols (SYN-RECV cache)
- attacker spoofs return address of many SYN handshake messages sent to victim
- victim's cache is filled (after ~20) pending incomplete sessions and delays are created



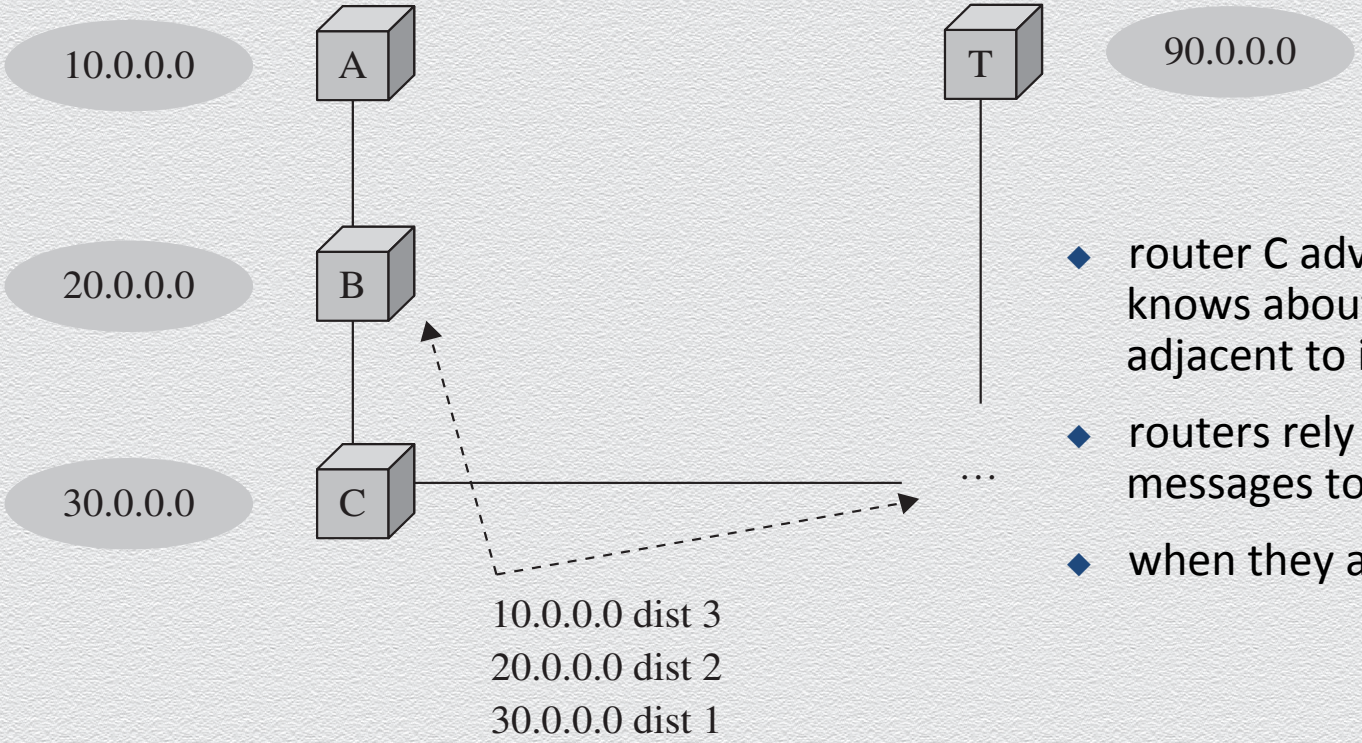
Teardrop Attack

IP datagrams allow to carry valiable-length data

- each datagram specifies the length of its data payload and its offset (start-end)
- the attacker can spoof these metadata so that recipient's state remains inconsistent
- sent packets cannot possibly be reassembled, as they conflict instructions
- in extreme cases, this can cause the entire OS to lock up



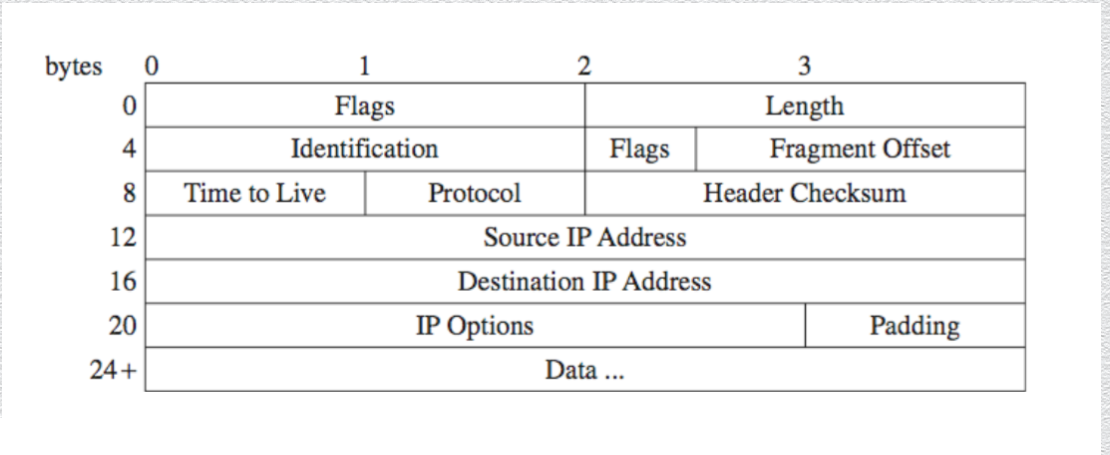
Rerouting routing

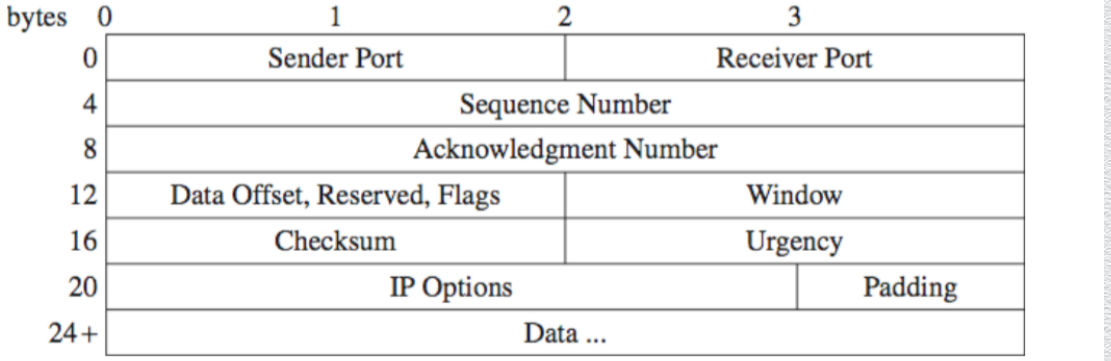


- router C advertises the routes it knows about to the routers adjacent to it
- routers rely on these advertising messages to be accurate
- when they aren't, DoS can ensue

TCP/IP headers

- headers of IP datagram and TCP packet
- IP: right
- TCP: below





Session hijacking

- an attacker is able to synchronize with a receiver while breaking synchronization with the sender and resetting the sender's connection
- the attacker continues the TCP session while the sender thinks the connection just broke off