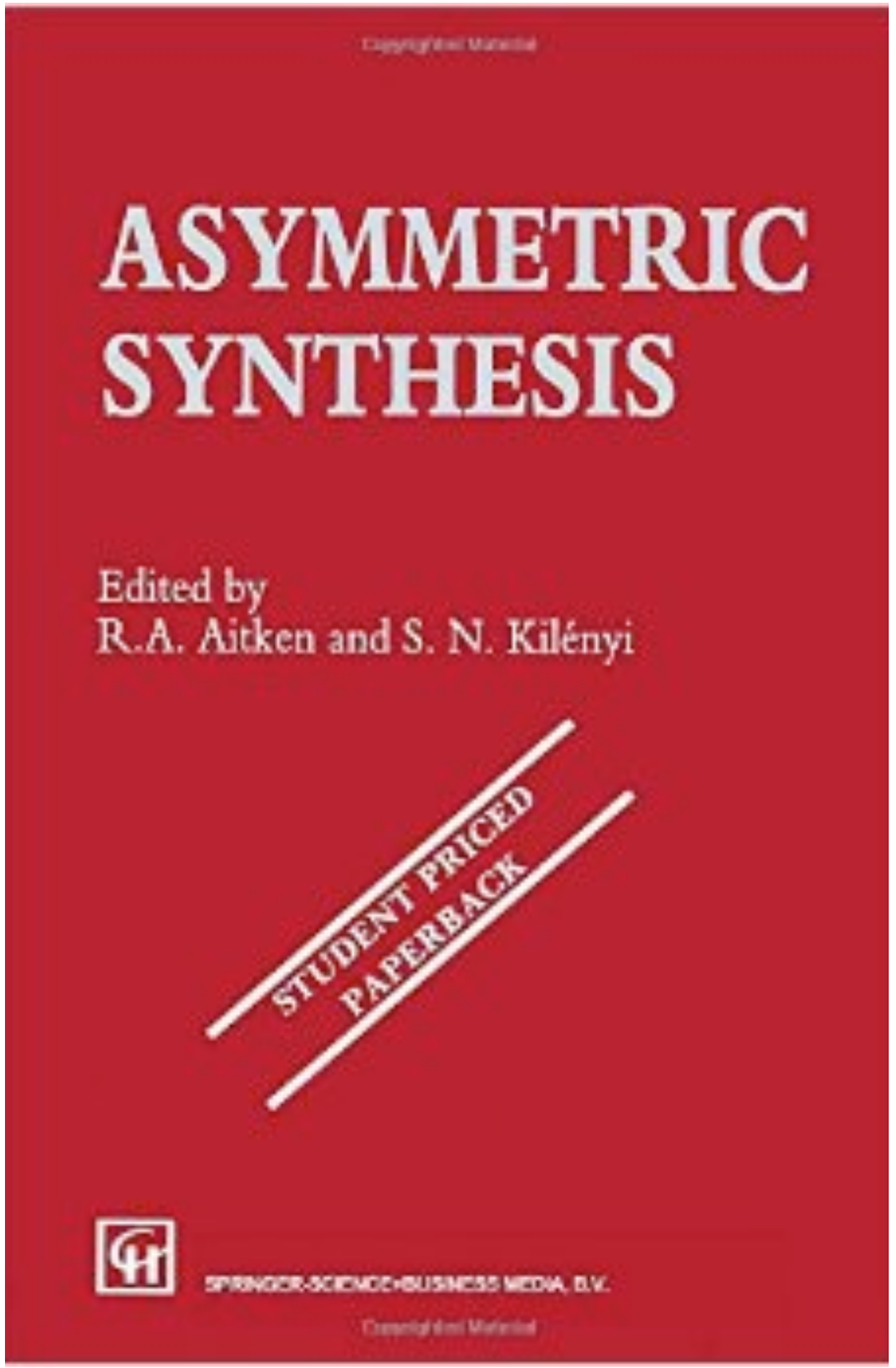
Asymmetric Encryption

With material from Jonathan Katz, David Brumley, and Dave Levin



A (very) little number theory

- Pick p, a random, **large** prime number
- g, a **generator** for p 1 < g < p if…
  - For all k between 1 and p:
    - There is some i, s.t. $k = g^i \bmod p$
- This is called ***discrete log***
  - Easy*:* Given p, g, x, compute $y = g^x \bmod p$
  - Believed **hard**: Given p, g, y, compute x
  - Candidate one-way function!

Generator examples: p = 7

g = 3

$3^1 \bmod 7$	3
$3^2 \bmod 7$	2
$3^3 \bmod 7$	6
$3^4 \bmod 7$	4
$3^5 \bmod 7$	5
$3^6 \bmod 7$	1

YES

g = 2

$2^1 \bmod 7$	2
$2^2 \bmod 7$	4
$2^3 \bmod 7$	1
$2^4 \bmod 7$	2
$2^5 \bmod 7$	4
$2^6 \bmod 7$	1

NO

Diffie-Hellman protocol

1 Compute $A = g^x \bmod p$

2 **p, g, A** →

Alice

secret value *x*

← **B**

Compute $B = g^y \bmod p$ 3

4

Bob

secret value *y*

$k = g^{xy} \bmod p$

5 $k = B^x \bmod p$

$k = A^y \bmod p$ 6

Successful secret key exchange!

Picking Large Primes

Sieve of Eratosthenes (10×10)

	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100

Primes:
2, 3, 5, 7,
11, 13, 17,
19, 23, 29,
31, 37, 41,
43, 47, 53,
59, 61, 67,
71, 73, 79,
83, 89, 97

Originally, use a **sieve**

We want **huge** primes

Better: Primality Tests

- Fermat's Little Theorem, if a not divisible by p
  - p prime implies a^(p-1) = 1 mod p
- Pick random a's, try the test
  - If a's powers of two, optimize w/ bit tricks
- Also, Miller-Rabin Primality Test
  - Probabilistic test for primality, run many times

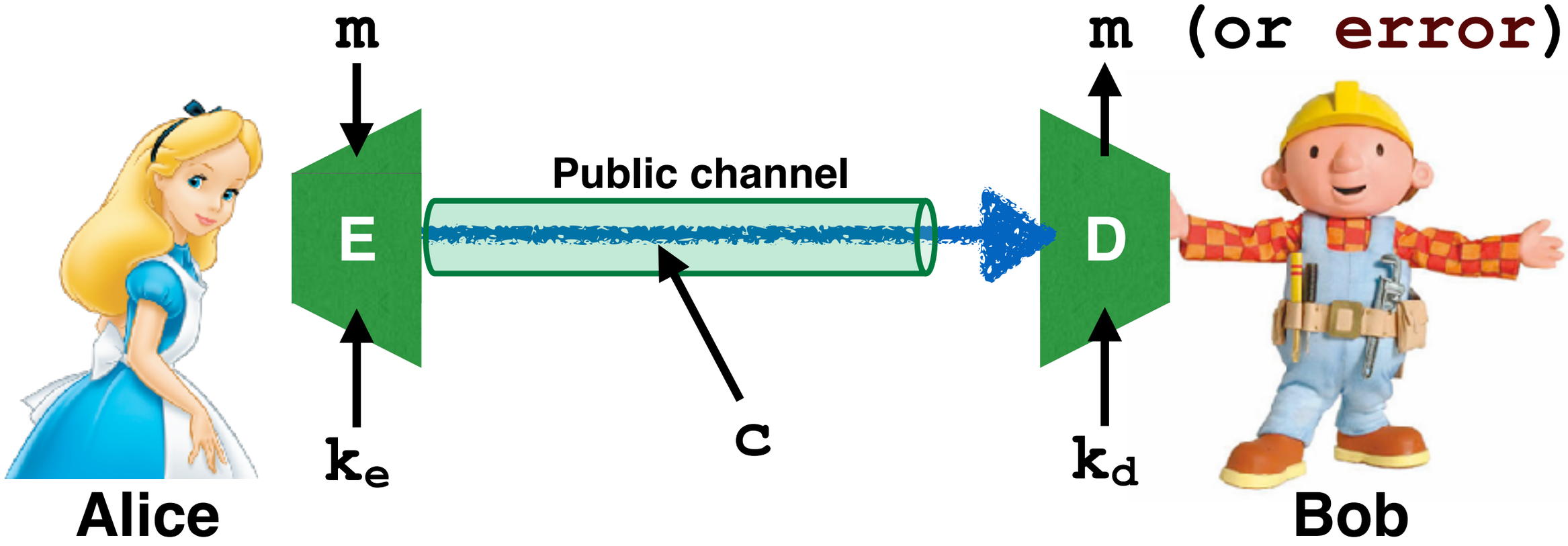
Public-Key Crypto

- Recall our three goals:
  - Confidentiality
  - Integrity
  - Authenticity

- Recall: Drawbacks of symmetric crypto
  - How to securely exchange keys?
  - Hard to scale
  - Limited authenticity / non-repudiation

We will use asymmetric crypto to mitigate these drawbacks!

Asymmetric crypto



- k_e != k_d
- k_d = **private** key, k_e = **public** key
  - Bob computes both, gives public key to Alice
- Alice sends a message to Bob: $c = E(m, k_e)$
- Bob can decrypt it: $m = D(m, k_d)$
- Anyone can send, **only Bob can read!**

Asymm. Cryptosystem: Definition

- Three polynomial-time algorithms:
  - KeyGen: Returns k_p (public) and k_s (secret)
  - *E(k_p,m)*: Encrypts *m* with k_p, returns *c* in *C*
    - Must be ***randomized*** (why?)
  - *D(k_s,c)*: Decrypts *c* with k_s, returns *m* in *M*
    - Or error

- Correctness condition:
  - For all pairs (k_p, k_s): D(k_s, E(k_p, m)) = m

Pros and Cons

- Scales well — everyone makes one key pair
  - Not *n* keys
- No direct setup comms between Alice and Bob
- Asymmetric is ***much, much slower***
- Asymmetric is easier to attack
  - Requires stronger assumptions

The authenticity problem

- In symmetric, we needed an ***authentic, private*** channel to exchange keys
  - Diffie-Hellman let us relax to ***authentic*** only
  - Public-key also requires authentic channel
- Who posted that ad in the NY Times?
  - Much more on this later

In practice: Hybrid

- Bob generates key pair and publishes k_p
- Alice generates new symmetric key k_{AB}
- Alice -> Bob: $c_1 = E(k_p, (\text{Alice} \parallel k_{AB}))$
- Alice -> Bob: $c_2 = E(k_{AB}, \text{message})$
- Arbitrary-length messages, efficiently
  - Keep k_{AB} as a session key

Intuition for algorithms

El Gamal (simplified)

- Similar to Diffie-Hellman
  - Public key: prime *p*, generator *g*, $h = g^x$
  - Private key: x, Alice **publishes** h and g

- Encryption: Sender chooses *y*
  - $c_1 = g^y$, $c_2 = m*h^y$

- Decryption: $m = c_2 / c_1^x$

- Security equivalent to D-H hardness

A teeny bit of number theory

- N = pq, where p and q are distinct primes
- $\phi(N) = (p-1)(q-1)$
  - Easy to compute if you know p and q; hard if not
- $a^b \bmod N = a^{b \bmod \phi(N)} \bmod N$
  - Take my word or take crypto
- Z_M^*: integers relatively prime to M
  - Have no common denominators except 1

Building to RSA (simplified)

- Choose *e* relatively prime to phi(N)
  - You can do mod arithemetic
- Choose *d* s.t. e*d mod phi(N) = 1
  - Easy if you know phi(N); else hard
    - By extension, easy if you know p and q
- Public key = (e, N); Private key = d

Textbook RSA

- Encrypt: $c = m^e \bmod N$
- Decrypt: $m = c^d \bmod N$
- Why does this work? $m^{ed} = m^1 = m$

Textbook RSA: NOT Secure

- Deterministic
- Leaks info about plaintext
- In practice: Preprocess message before applying RSA permutation
  - Randomized padding, hash permutations

PKCS #1 v1.5

- You need 1024 total bits
- Pad message: $c = (r \,||\, m)^e \bmod N$
  - r is (mostly) a random number
- Check padding on decryption to detect error

Is RSA hard?

- Easy to compute m when we know d (of course)
  - But what about if we don't?
- Challenge: Compute x given $c = m^e \bmod N$
  - Easiest known way: Factor N into p and q
    - Believed (not proven) nothing easier
  - Factoring N is believed hard (but not proven)

How hard is hard?

- Best current algorithms to factor N=pq
  - p and q equal-length
  - runs in $\approx \exp(|N|^{1/3})$

- Currently |N| ~ 1024 for OK security
  - ~2048 to be sure

How hard is hard?

- World record: RSA-768 (232 digits)
  - Two years, hundreds of machines
  - Equivalent to 2000 single-core years!

- Factoring 1024-bit integer
  - About 1000 times harder
  - …. Possible this decade?

Implementation attacks

- Timing and power:
  - How long / how much to compute c^d mod N
- Bad randomness:
  - p and q can’t be predictably generated
  - If N = pq and N’ = pq’, both are broken
- Bad padding / malleability

Malleability

- Given c (m unknown), can construct c’ that will decrypt to a related message m’
  - Recall CBC attack last time

Recall:

CBC is not CCA-secure

Challenge:
Choose b = x or y
at uniform random

— — —

Decryption oracle

m_x and m_y

$|m_x| = |m_y| = 1$ blk

$c = E(k,m_b) = IV \,||\, c[0]$

$c' = (IV \text{ xor } 1) \,||\, c[0]$

$m' = D(k,c') = m_b \text{ xor } 1$

Eve
(polynomial time)

Uh oh.

Malleability

- Given c (m unknown), can construct c' that will decrypt to a related message m'
  - Recall CBC attack last time
  - CBC, CTR are malleable; auth. encr. is not!
- Basic El Gamal and basic RSA are malleable
  - CCA-safe variations exist

Bleichenbacher attack

- Insecure padding, malleability
  - Return error if padding not formatted correctly
- Allows gradual CCA attack based on error detection
  - Analogous to blind ROP attack?

In practice

- Need CCA security for real applications
- Symmetric: Use authenticated encryption
- Use approved pub key scheme
- Hybrid: Combine!
  - Secure if components are

Digital signatures

Signatures for integrity

- Sign with your private key
- Anyone can verify using public key
  - Assuming private key is secret, only you could have sent the message
- e.g., Sign software patches
  - Public key bundled with initial software

Signatures vs. MACs

Manage one key	Manage n keys
Sign once, verifiable by anyone	Sign separately per verifier
Public non-repudiation	Nope

Defining a signature scheme

- Keygen: outputs k_p and k_s
- $s = S(k_s, m)$
- $V(k_p, m, s)$ outputs true or false
- Correctness:
  - For all pairs (k_p, k_s): $V(k_p, m, S(k_s, m)) = true$

Signature security game

- No existential forgeries (analogous to MAC)

Signing oracle

— — —

Verify s' or error

$m_1 \ldots m_n$

$s_1 = S(k_s, m_1) \ldots s_n$

m', s' (not in m_i)

Eve
(polynomial time)

Security IFF $Pr[V(k_p, m', s') = 1]$ is very small!

Naive RSA signatures

- Public key (e,N) and private key (d,N)
  - Recall: e*d ~ 1 (mod arithmetic)
- $s = m^d \bmod N$
- Verify whether $s^e \bmod N = m$
- This is ***easily existentially forgeable***
  - Choose s. Calculate m.

RSA signatures (better)

- Send s = $H(m)^d$ mod N along with m
  - Use a good cryptographic hash function H
- Recipient calculates digest g = s^e mod N
  - Verify g == H(m)
- Why does this fix the problem?
  - You can choose s' and find the matching digest g'
  - BUT, preimage resistance means that you can't pick a message m' s.t. g == H(m')
- Variants of this approach are believed secure
  - Assuming RSA is hard
  - Bonus: Handles long messages "for free"