Ordinary Math in Everyday Life

Jesse Walker, Ph.D.

Intel Corporation – Intel Labs – Security and Privacy Research

jesse.walker@intel.com

Academia v. Industry

Academic math is like fine dining …

Industrial math is more of a smorgasbord …

Agenda

- Wi-Fi
- Random Number Generation
- Anonymous Authentication

WEP: the original Wi-Fi encryption

Wi-Fi

IEEE 802.11b, the original Wi-Fi specification, was published in 1999

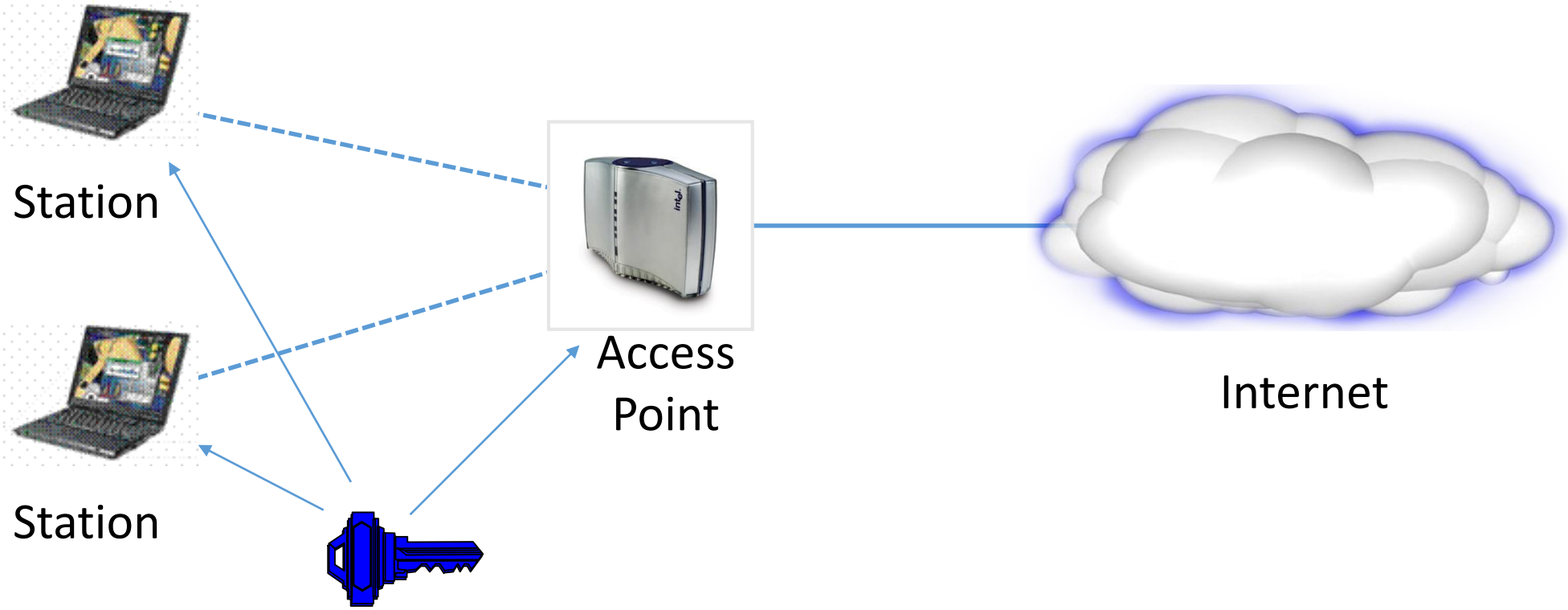
Wi-Fi used **W**ired **E**quivalent **P**rivacy for encryption

- Or **WEP** for short

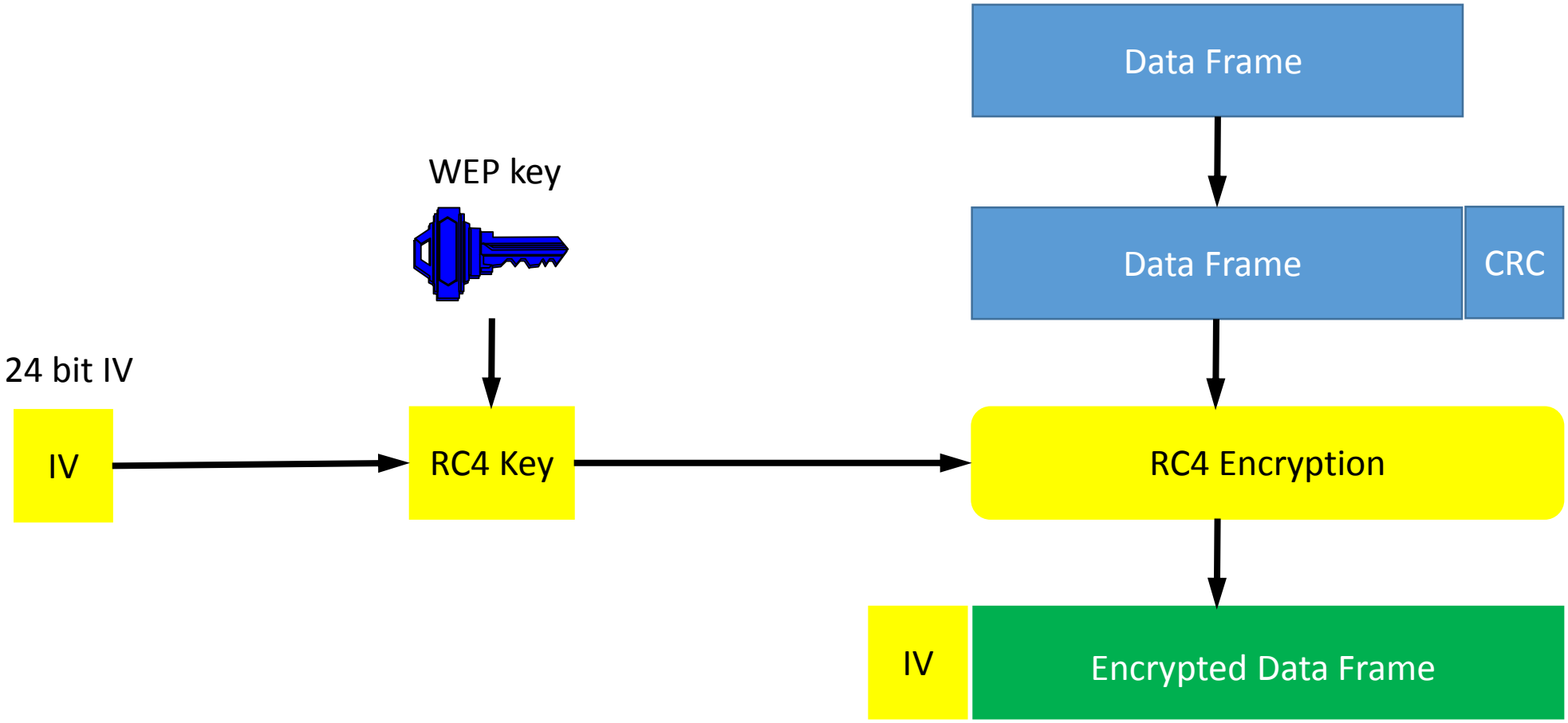
One day in 2000 Duncan Kitchin, then Vice Chairman of the IEEE 802.11 Working Group, walked into my office

- Duncan: <describes 802.11 authentication> “Does 802.11 authentication work?”
- Jesse: “No; this is a bad design. Is there anything else to know?”
- Duncan: “Here is the spec.”

How WEP works



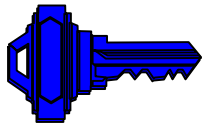
How WEP works



How RC4 encryption works

RC4 is a **stream cipher**

encryption key
= $IV \,||\, RC4\ key$



Pseudo-Random Number Generator

1 byte k of **key stream**

1 byte p of **plaintext data** → + → 1 byte c of **ciphertext data**, $c = k + p$

+ = addition in the vector space $(F_2)^8$

Fundamental property of addition in $(F_2)^n$: $k + (k + p) = p$

Decryption generates same key stream byte k, adds it with ciphertext byte c to recover the plaintext byte p

Thought experiment

What happens if the same key stream byte k is used to encrypt different plaintext bytes p_1 and p_2?

- $c_1 = k + p_1$ and $c_2 = k + p_2$

Answer: Then $c_1 + c_2 = (k + p_1) + (k + p_2) = p_1 + p_2$

- If the attacker knows p_1, then he can recover p_2 without knowing the key stream byte k or the encryption key

Hence, a stream cipher implies we must **NEVER** reuse the same key stream to encrypt different plaintext

- For WEP, we must have a new IV for every frame or else change the encryption key

WEP cryptanalysis

How many frames before key must be changed?

- The IV is 24 bits $\Rightarrow$ the key must be changed after at least 2^{24} frames

The number and identities of devices can change as client roam

The most reasonable strategy picks the WEP IV randomly

- If $N = 2^{24}$, the birthday problem says there should be a collision after about $2^{12} \approx 4000$ frames
- 4000 frames $\approx$ 1 second on a busy LAN, about 20 seconds on a normal LAN

WEP keys must be manually reconfigured – oops!

Polynomials and message representation

Often useful to represent messages as polynomials:

- Represent the s-bit message M as a sequence of bits

$$M = m_0\, m_1 \ldots m_{s-2}\, m_{s-1}, \text{ where } m_i \in \{0,1\}$$

- Each bit of M can be considered the coefficient of the polynomial

$$M(X) = m_0 X^{s-1} + m_1 X^{s-2} + \ldots + m_{s-2} X^1 + m_{s-1}, \text{ where } m_i \in F_2$$

Let $F_2[X]$ denote the polynomials in X with coefficients in F_2 = the field with two elements $\{0,1\}$

Then every message M can be considered a polynomial in $F_2[X]$

Cyclic redundancy check (CRC)

Choose an **irreducible** polynomial $p(X) \in F_2[X]$ of degree n

For any $G(X) \in F_2[\mathrm{X}]$ an ***n*-cyclic redundancy check** (**CRC**) is the $n-1$ degree polynomial $g(X)$ satisfying $g(X) = G(X) \pmod{p(X)}$

- $\mathrm{crc32}(X) = X^{32}+X^{26}+X^{23}+X^{21}+X^{20}+X^{16}+X^{12}+X^{11}+X^{10}+X^{7}+X^{5}+X^{2}+X+1$ is the irreducible polynomial used by WEP

Shannon introduced CRCs to detect random bit errors on communications channels

- An n-CRC can detect up to n random bit errors on the channel
- Not designed to detect malicious errors

How WEP works redux

- An s bit message M corresponds to the polynomial

$$M(X) = m_0X^{s-1} + m_1X^{s-2} + \ldots + m_{s-2}X^1 + m_{s-1}$$

- WEP step 1: append a 32-CRC based on an irreducible polynomial $\mathrm{crc32}(X)$ to $M(X)$ prior to encryption:

$$X^{32}\cdot M(X) + g(X) =$$

$$(m_0X^{s+31} + m_1X^{s+30} + \ldots + m_{s-1}X^{32}) + (g_0X^{31} + g_1X^{30} + \ldots + g_{31})$$

  where $g(X) = M(X) \pmod{\mathrm{crc32}(X)}$ and $g(X) = g_0X^{31} + g_1X^{30} + \ldots + g_{31}$

- WEP step 2: encrypts by adding a key stream polynomial $K(X) = k_0X^{31+s} + \ldots + k_{s+31}$:

$$\mathrm{WEP}(M) = K(X) + (X^{32}\cdot M(X) + g(X)) =$$

$$(k_0+m_0)X^{s+31} + (k_1+m_1)X^{s+30} + \ldots + (k_{s-1}+m_{s-1})X^{32} + (k_s+g_0)X^{31} + (k_{s+1}+g_1)X^{30} + \ldots + (k_{s+31} + g_{31})$$

Thought experiment 2

We can change bit m_i of the encrypted message WEP(M) by adding X^{i+32} to WEP(M), but then the CRC is wrong after decryption

Idea: add $i(X)$, where $i(X) = X^{i+32} \pmod{\text{crc32}(X)}$, to WEP($M$) as well!

- Change

$$\text{WEP}(M) = K(X) + (X^{32} \cdot M(X) + g(X))$$

to

$$\text{WEP}(M) = K(X) + (X^{32} \cdot M(X) + \boldsymbol{X^{i+32}} + g(X) + \boldsymbol{i(X)})$$

- This decrypts to $M(X) + X^i$ and $\text{crc32}(M(X) + X^i) = g(X) + i(X)$

We can forge WEP messages by bit flipping and patching the CRC

Random number generation

Randomness' role in crypto

Implicit Expectation: "Secure" systems work as specified, independent of what the environment (i.e., any attacker) can do to it (i.e., without any constraints on the environment)

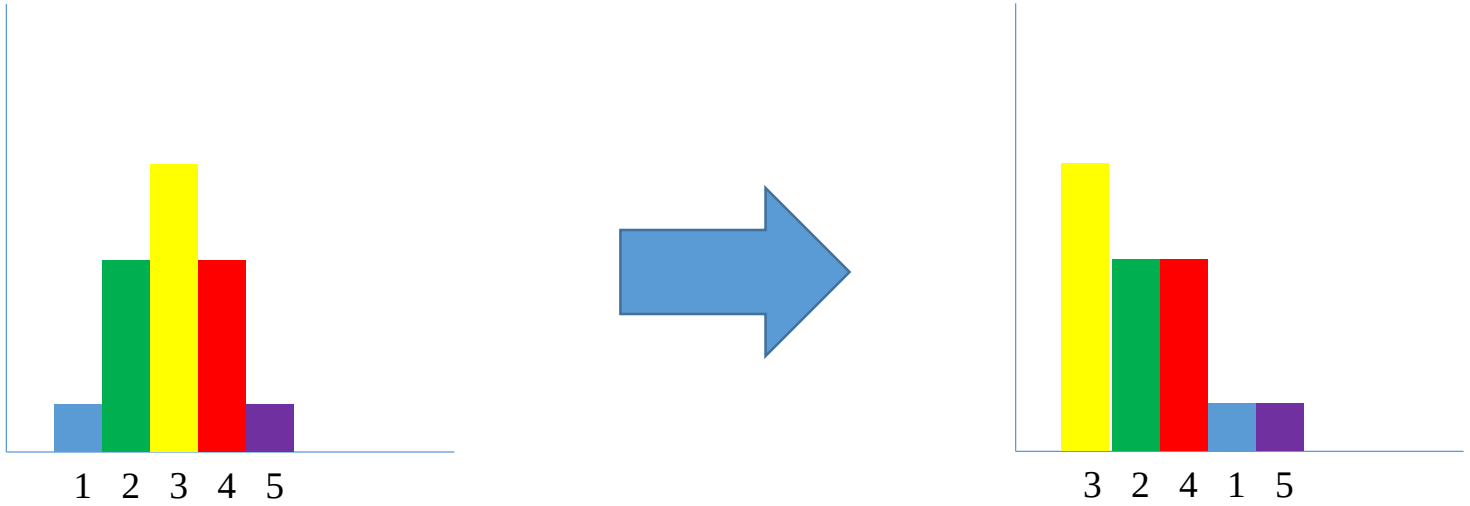
Question: How can we defeat ***ALL*** computationally bounded adversaries?

- Even the ones we haven't thought about?

Strategy: Use **randomness** to wall off attack below a computational complexity threshold

- Crypto algorithm designers embed $O(2^n)$ random search problems into its designs
- If n is sufficiently large, then $O(2^n)$ operations is beyond anyone's computational resources

Crypto’s randomness must be perfect



Deviation from uniform decreases the attacker’s work

Traditional approach to RNGs

Find an entropy source in nature

- Johnson thermal noisy
- Radioactive decay

Engineer the source and its sampling method to make the output as close to uniform as possible

This has always failed in practice

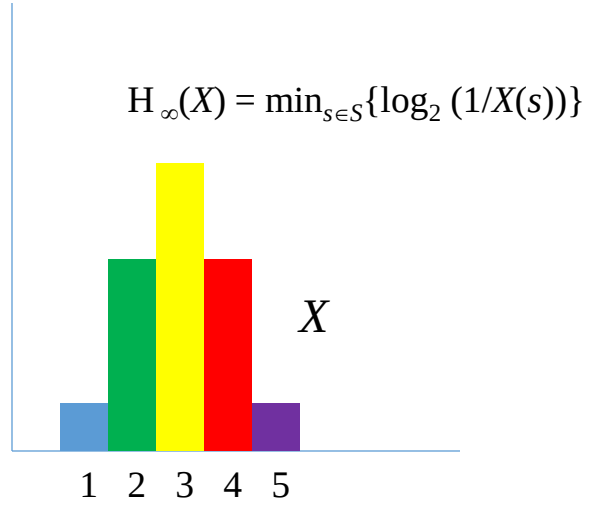
- Our engineered sources age – silicon characteristics change over time
- Our engineered sources exhibit different behavior across environmental conditions – as thermal and electrical conditions change, so does the source and the sampling method

How is Randomness Represented?

- A **random variable** $X : S \to \mathbb{R}$ models measurements of some random process
- The **information** of a random variable X is itself a random variable defined as $-\log_2(X) = \log_2(1/X)$
  - The information $\log_2(1/X(s))$ says how many bits are needed to unambiguously represent state s
  - If the number of bits of $X(s)$ exceeds $\log_2(1/X(s))$, then X contains **redundant** information
- The **entropy** $\mathrm{H}(X)$ of a random variable X is the negative of the expected value of X's information: $\mathrm{H}(X) = \mathbf{E}_X(-\log_2(X)) = \Sigma_{s \in S}\, X(s) \cdot \log_2\,(1/X(s))$
  - The entropy measures the randomness or unpredictability of X in bits
- The **min-entropy** is $\mathrm{H}_\infty(X) = -\min_{s \in S}\{\log_2\,(X(s))\}$
- $\mathrm{H}_\infty(X) \leq \mathrm{H}(X)$, with equality if and only if $X(s) = 1/|S|$ for all $s \in S$
  - Every sample from X has at least bits $\mathrm{H}_\infty(X)$ bits of entropy

Example

s	$X(s)$	$\log_2(1/X(s))$	$X(s)\cdot\log_2(1/(X(s))$
1	1/16	4	1/4
2	1/4	2	1/2
3	3/8	$3 - \log_2(3) \approx 1.415$	$3(3 - \log_2(3))/8 \approx 0.531$
4	1/4	2	1/2
5	1/16	4	1/4



$H(X) = E_X(\log_2(1/X)) = \Sigma_{s\in S}\, X(s)\cdot\log_2(1/X(s)) \approx 1/4 + 1/2 + 0.531 + 1/2 + 1/4 = 2.031$, so

$H_\infty(X) = -\min_{s\in S}\{\log_2(X(s))\} = \log_2(\min_{s\in S}\{1/(X(s)\}) = 3 - \log_2(3) \approx 1.415$

Every sample of X has at least $H_\infty(X) = 1.415$ bits of entropy

The Privacy Amplification Problem

Alice

Alice and Bob share a 2000 bit secret key K to secure their communication against their arch-nemesis Eve

Bob

They learn that Eve has learned part of K, say 200 bits . . .

. . . but they don't know which 200 bits

Alice and Bob know:

$$H_\infty(K) = 2000 - 200 = 1800$$

Is there some way they can still use K?

Privacy Amplification Solution

The **Leftover Hash Lemma** of Impagliazzo, Levin, and Luby (1989) solves the privacy amplification problem

- Definition. A family $\mathcal{H}$ of functions $h : S \to \{0,1\}^n$ is **ε-universal** if for all $s, t \in S$

$$\Pr_{h \in \mathcal{H}}[h(s) = h(t)] \leq \varepsilon$$

- Theorem (Leftover Hash Lemma). Assume $\mathcal{H} = \{h : X \to \{0, 1\}^n\}$ is a $(1+\eta)/2^n$-universal hash family. Then if h is selected uniformly over $\mathcal{H}$ then

$$\sum_{s \in S} |h(X(s)) - U_n(X(s))| \leq (\eta + 2^n/2^m)^{1/2}/2$$

  where $\mathrm{H}_\infty(X) \geq m$

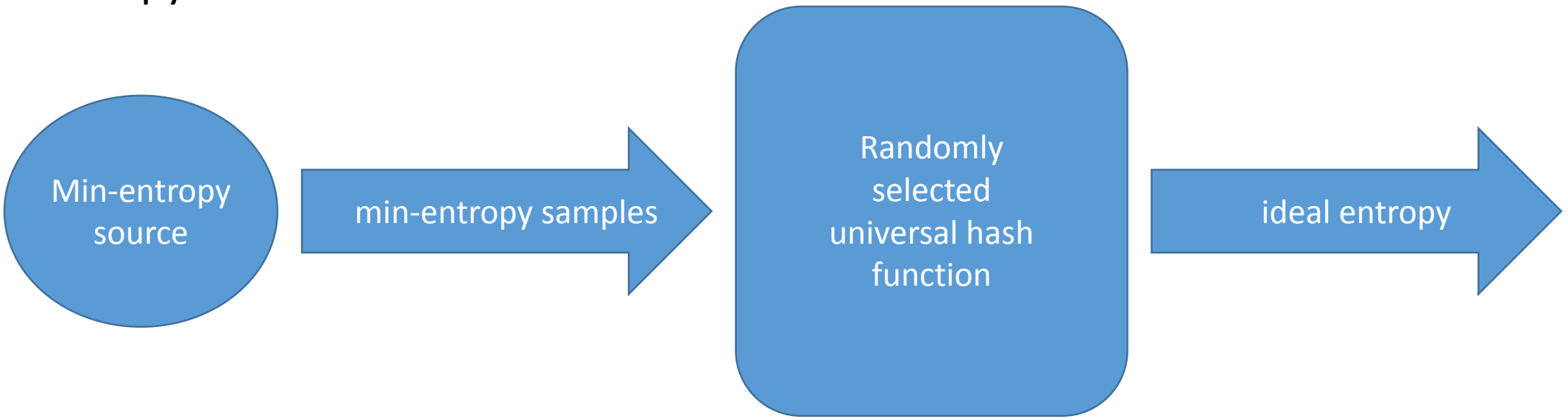
  - U_n denotes the uniform distribution on $\{0,1\}^n$

Translation: universal hash families are efficient entropy extractors

Central Idea

Even though ideal entropy sources are hard to find in nature

- We may still hope to find sources that produce significant amounts of entropy, i.e., find X with $\mathrm{H}_\infty(X) \geq m$
- If an entropy source X satisfies $\mathrm{H}_\infty(X) \geq m$ for some $m > 0$, then we can apply the Leftover Hash Lemma to extract indistinguishable from ideal entropy from min-entropy



Requirements

Want a source that can be faithfully modeled

- Must be simple enough to model with a random variable X
- X must admit a computable min-entropy
- If the entropy source empirically acts like X there is no reason to doubt it has the min-entropy of X

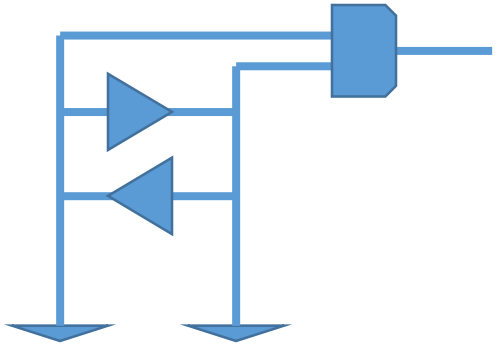
All digital, no analog components

- No redesign and revalidation for new process technologies

Produce bits at a rate directly useful to applications

- e.g., at least 100 Mbps for argument's sake, not 75 Kbps

Intel’s entropy source



Invented by Intel hardware engineer Charles Dike

It is latch built from a pair of cross-coupled inverters

- Circuit assumes two stable (0/1) and one unstable state (meta-stable)
- At power-on circuit enters the meta-stable state
- Circuit held in meta-stable state until Johnson thermal noise resolves circuit’s value to 0 or 1
- After the circuit resolves and outputs one bit value, power it off
- Repeat at device clock rate

Entropy source model

We modeled our source as an Ornstein-Uhlenbeck stochastic process

- The Ornstein-Uhlenbeck process is the only stationary, Gaussian, and Markovian process
  - It models a mean-reverting random walk
- A digital latch tends to resolve to its previous state, so our circuit slightly biases the next output to be different from the previous
- Has a computable min-entropy

Anonymous authentication

Signature schemes

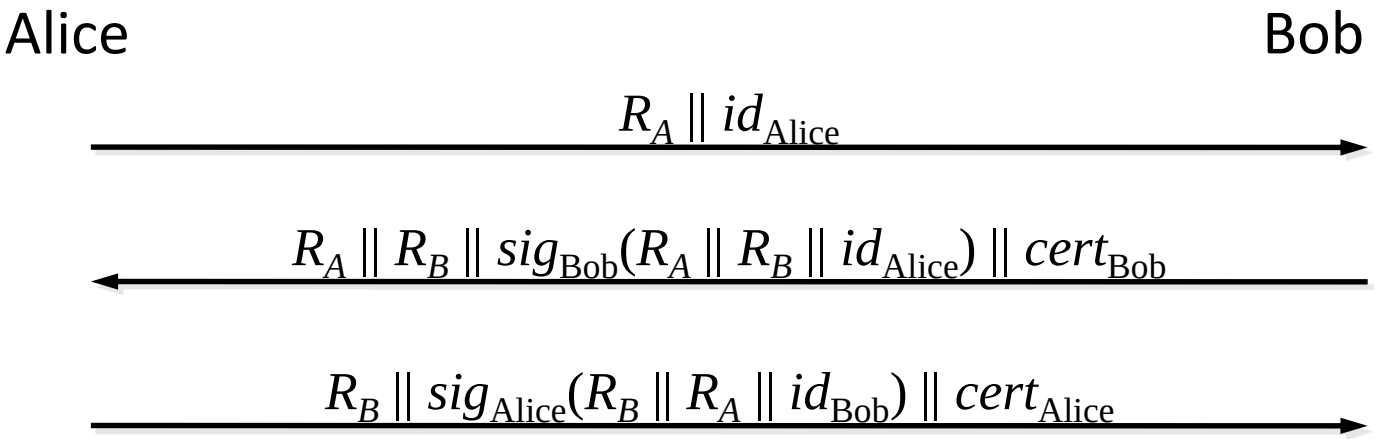
A **signature scheme** is a collection of three algorithms

- $KeyGen(k)$: produces a pair of k bit keys (sk, pk) for the scheme
  - sk is called the **secret** or **private key**, pk the **public key**
- $Sign(sk, msg)$: for any message msg produces a **signature** σ using sk
- $Verify(pk, msg, \sigma)$: returns TRUE if σ was produced by the $Sign$ operation using sk and msg and FALSE otherwise

Example: RSA

- $KeyGen(2n)$: Choose independent randomly generated n bit primes p, q, and choose $0 < e < \varphi(pq) = (p-1)(q-1)$ satisfying $\gcd(e, \varphi(pq)) = 1$, $sk = e$, $pk = (N, d)$, where $d = e^{-1} \bmod \varphi(pq)$ and $N = pq$
- $Sign(sk, msg)$: $m \leftarrow hash(msg)$, $m' \leftarrow pad(m)$, $\sigma \leftarrow (m')^e \pmod N$
- $Verify(pk, msg, \sigma)$: $mm \leftarrow \sigma^d \pmod N$, $mm' \leftarrow unpad(mm)$, **return** $(mm' = hash(msg))$?

Conventional authentication protocols



Alice commits to her identity (and to her signing key key) in message 1

This commitment binds message 3 to message 1

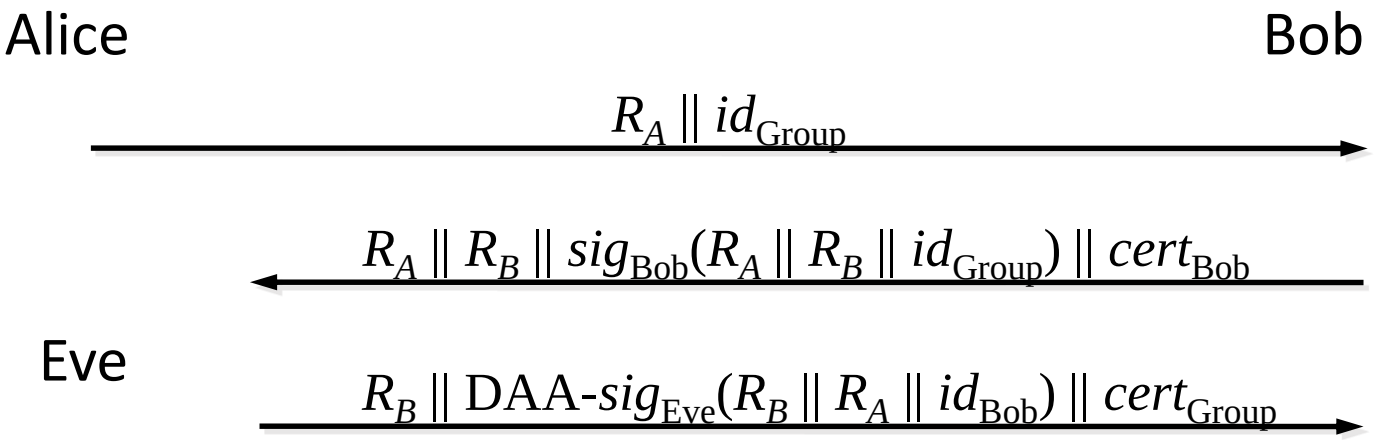
DAA

In 2004 my Intel colleague Ernie Brickell invented **D**irect **A**nonymous **A**ttestation (**DAA**)

- A new type of signature scheme
- All group members share the same public key but have distinct secret keys
- A DAA signature does not reveal which group member created it, only that some group member created it

DAA promise privacy preserving authentication

Attempt 1 to use DAA



Oops

What can replace identities to provide the binding function when using DAA?

What should “secure” mean?

Model for reasoning about Protocols

The players

- A group consisting of principals $P_1, \ldots, P_n$, each with a DAA signature scheme for the group
- Each principal P_i is represented in instance s of the protocol by an oracle $O_{i,s}$
- A verifier Q ($Q = P_i$ for some i) using a classical signature scheme
- An oracle **accepts** and outputs a session descriptor (P,s,Q) if the oracle's protocol instance s completes successfully
- An **network-adversary** $\mathcal{A}$ is the environment through which principals and their oracles interact

Modeling the adversary's capabilities

- A network-adversary $\mathcal{A}$ interacts with the principals/oracles through queries
  - *Send*(P,Q,s,m): P's oracle $O_{P,s}$ sends message m to Q's oracle $O_{Q,s}$
  - *Session-Key-Reveal*(P,Q,s): P's oracle $O_{P,s}$ gives its session key sk to $\mathcal{A}$
  - *State-Reveal*(P,s): P's oracle $O_{P,s}$ gives its entire session state to $\mathcal{A}$
  - *Corrupt*(P): P and all its oracles give their internal state to $\mathcal{A}$
  - *Expire-Session*(P,Q,s): P's oracle $O_{P,s}$ deletes its state
  - *Test*(P,Q,s): P's oracle $O_{P,s}$ randomly chooses a bit b. If $b = 1$ $O_{P,s}$ gives $\mathcal{A}$ its session key; otherwise $O_{P,s}$ gives $\mathcal{A}$ a randomly generated string. Used at most once by $\mathcal{A}$

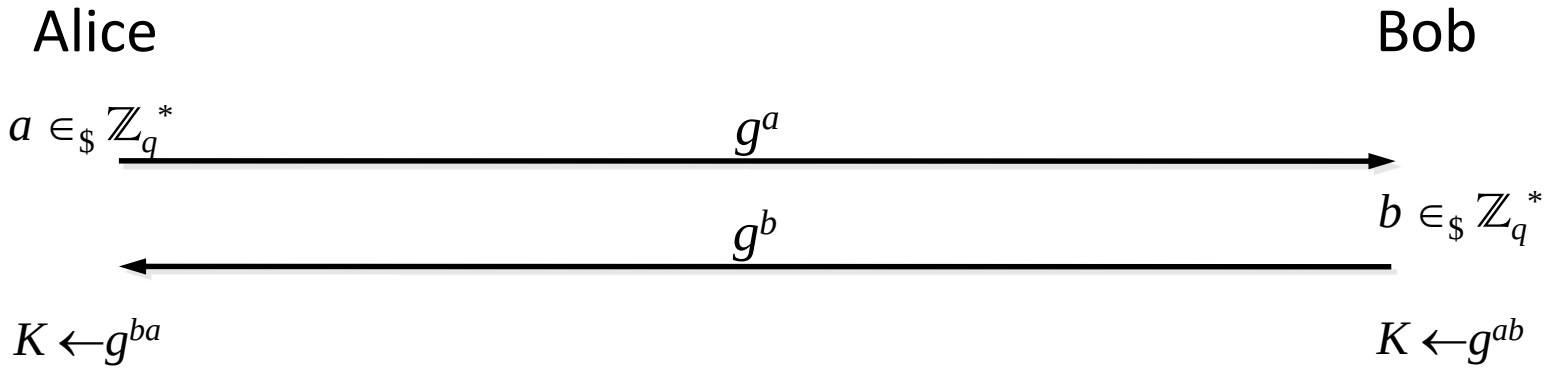
Security

- Definition 1. Suppose an oracle $O_{P,s}$ has accepted with output (P,s,Q). The oracle $O_{Q,s}$ is the **matching session** if
  - $O_{Q,s}$ has accepted with output (Q,s,P) or
  - $O_{Q,s}$ has not completed the execution of the session
- Definition 2. A protocol π is **secure** if for all probabilistic polynomial time network-adversaries $\mathcal{A}$ the following hold
  - If two uncorrupted parties P and Q complete matching sessions $O_{P,s}$ and $O_{Q,s}$ with outputs (P,s,Q) and (Q,s,P), then the corresponding session keys are the same except with negligible probability
  - $\mathcal{A}$ succeeds in distinguishing the output from its $Test$ query with probability no more than ½ plus a negligible function

Diffie-Hellman

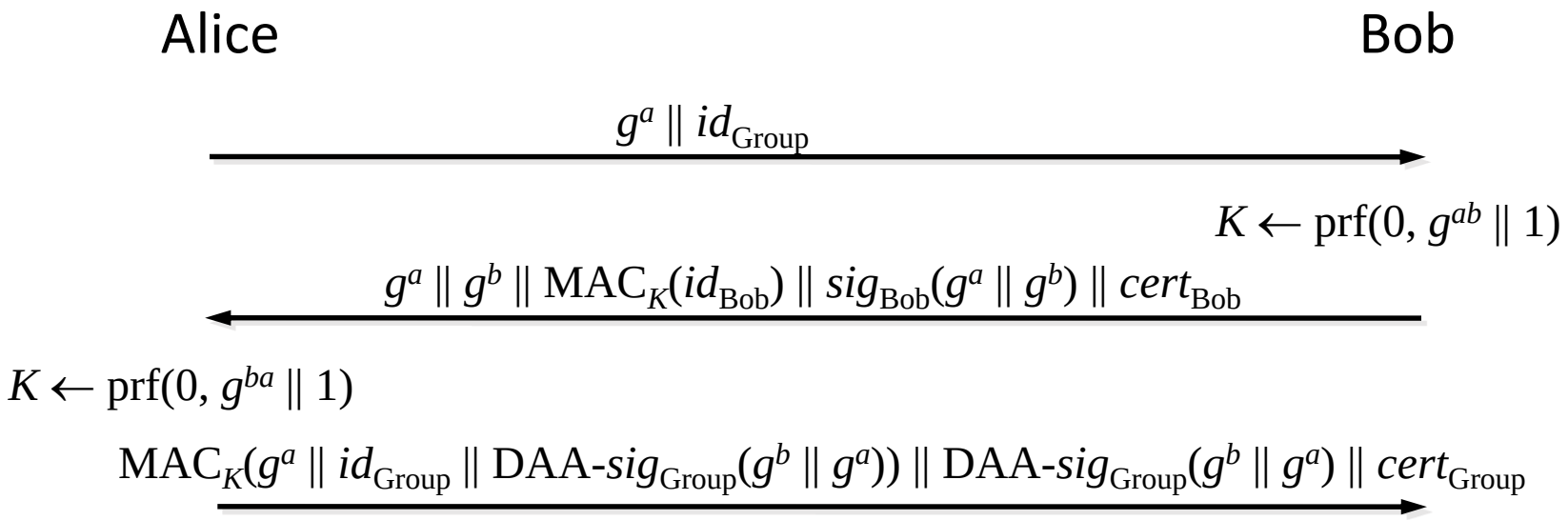
Let G be an Abelian group with

- A cyclic subgroup $<g>$ of prime order q in which the **Decision Diffie-Hellman problem** is hard, i.e., it is computationally intractable to distinguish (g^a, g^b, g^{ab}) and (g^a, g^b, g^c), c random
- e.g., the points on an elliptic curve $y^2 = x^3 + sx + t$ with $s \neq 0$ over a finite field F_p, with p a well-chosen prime



Bob knows K can only be computed by itself and the party that knows a (which Bob doesn't know)
Alice knows K can only be computed by itself and the party that knows b (which Alice doesn't know)

The DAA-Sigma protocol



This protocol uses **Diffie-Hellman** for commitment and proof

Theorem

Theorem (Walker-Li). Let G be a group in which the DDH assumption is true, and suppose the DAA signature scheme, PRF, and MAC are secure. Then DAA-Sigma is secure against any probabilistic polynomial time network-adversary.

Idea behind proof:

- If the protocol does not meet the definition of secure, then an adversary $\mathcal{A}$ exists that can cause two uncorrupted parties to disagree about an unrevealed session, and we can use $\mathcal{A}$ to design an algorithm breaking one of the underlying primitives:
  - The signature scheme
  - The PRF
  - The MAC or
  - Diffie-Hellman
- If the underlying primitives are secure, this is a contradiction

DAA-Sigma is part of TPM 2.0 and ISO 20009

Summary

- Math is everywhere and is varied as life itself
- Theory is a good guide to practice
- Proof is still needed in the real world
  - — Good definitions lead us to good algorithms

Feedback?