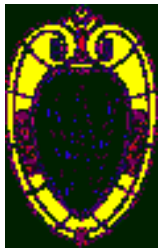


# CS/ECE/Math 435
# Introduction to Cryptography

*Professor Chris DeMarco*
*Department of Electrical & Computer Engineering*
*University of Wisconsin-Madison*

*Spring Semester 2021*

# MD4 Hash Function Architecture

- State space is equivalent to $\mathbb{Z}_2^{128}$, but is organized as $\mathbb{Z}_2^{32} \times \mathbb{Z}_2^{32} \times \mathbb{Z}_2^{32} \times \mathbb{Z}_2^{32}$ (i.e. state $\underline{q}$ is set of four 32-bit strings or binary vectors).

- Message blocks $\underline{x}_i \in \mathbb{Z}_2^{512}$ (512-bit strings or binary vectors). Again, recall original $\underline{x}$ is of arbitrary length "*," yielding $l$ blocks of length 512.

- The design philosophy of multiple "rounds" shares alot with DES: 16 rounds of nonlinear function applications are used.

Observe: $\underline{\ell} \in (\mathbb{Z}_2^{32})^4$, i.e.

$\mathbb{Z}_2^{128}$ partioned into 4 32-bit strings.

Likewise, a 512-bit block of input partitioned in 16 32-bit strings, ONE FOR EACH ROUND.

So . . .

$\underline{q}_i \in \mathbb{Z}_2^{128}$ : state at "time" index $i$

partitioned into

$A_i, B_i, C_i, D_i$ each $\in \mathbb{Z}_2^{32}$

---

$\underline{X}_i \in \mathbb{Z}_2^{512}$ : $i^{th}$ block of original message, $\underline{X} \in \mathbb{Z}_2^{*}$.

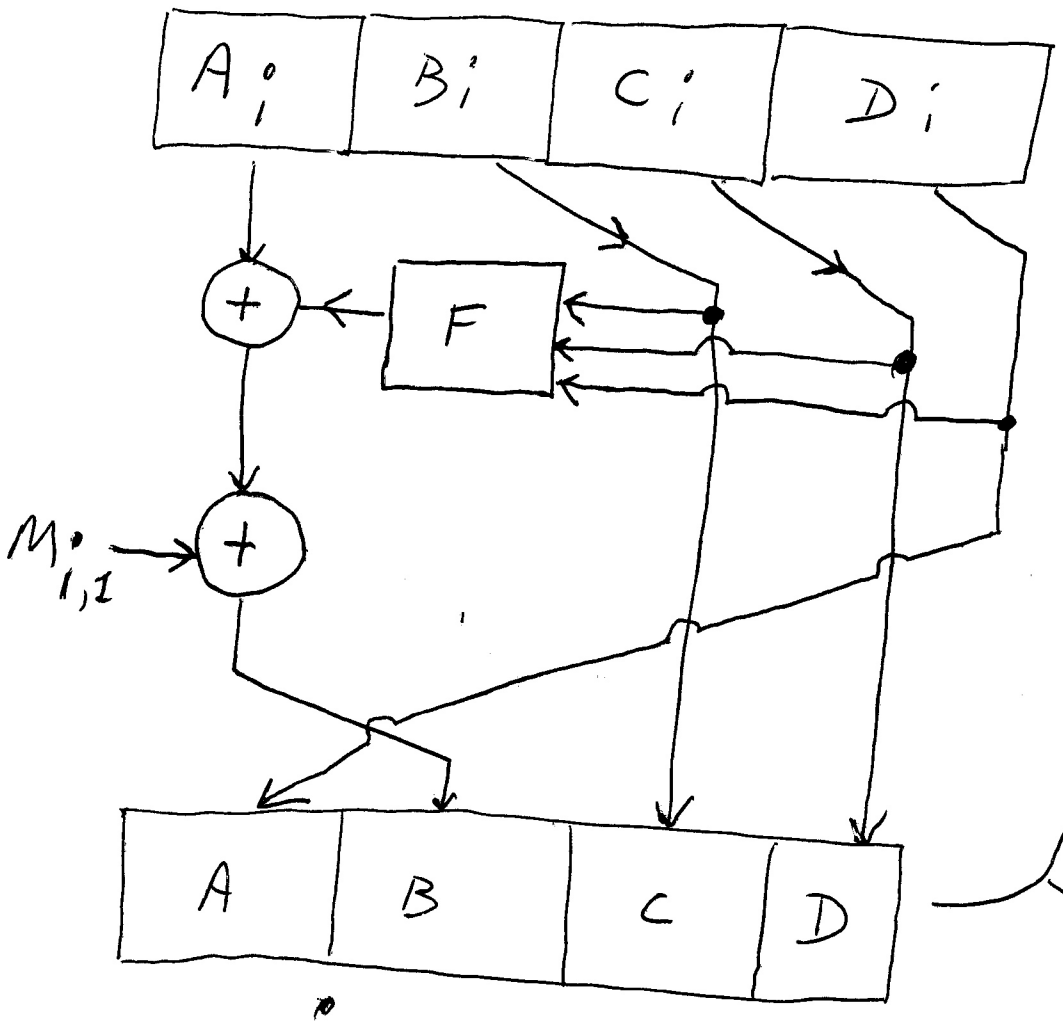
This $X_i$ is further partitioned into 16 subblocks,

$M_{i,1}, M_{i,2} \ldots M_{i,16} \in \mathbb{Z}_2^{32}$

- With these partitions, all computational steps operate on 32-bit strings / binary vectors

- On following page's block diagram, $\oplus$ denotes component-wise addition (mod 2) of vectors in $\mathbb{Z}_2^{32}$

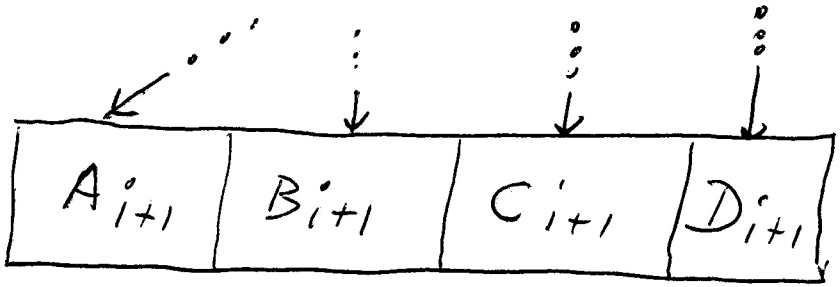
- Exact nature of F not detailed here; simply observe F is nonlinear Boolean function $F: \mathbb{Z}_2^{32} \times \mathbb{Z}_2^{32} \times \mathbb{Z}_2^{32} \to \mathbb{Z}_2^{32}$




← state $Q_i$ at time index $i$

round 1 intermediate result, "on-the-way" to producing $Q_{i+1}$

repeat 15 more rounds w/ $M_{i,2}, M_{i,3} \ldots$




round 16 final result; this becomes $Q_{i+1}$

While useful to have a general overview of design philosophy / architecture of MD4, this circa-1990 design is far from state of the art. Like DES, attacks on MD4 are well-known, and well within computational power of circa-2021 computing.

(So, ok for applications of modest security requirements; good security practice uses variants of SHA-2 standard)

At each time $i$, we update state:

$$\underbrace{q_{i+1}}_{\text{new state}} = \delta(\underbrace{q_i}_{\substack{\text{old state} \\ \text{at time } i}}, \overbrace{x_i}^{\substack{\text{input (part of} \\ \text{message } x) \text{ at} \\ \text{time } i}})$$

Evaluation of $y = h(x)$:

1) Partition $x$;

2) Run finite state machine, initialized at time $i = 0$, to final time that yields $q_\ell = \delta(q_{\ell-1}, x_{\ell-1})$

3) Let $y = q_l \in Z_2^n$

General Observations

- Computing $h$ requires $l$ forward evaluations of $\delta$; because

$\delta : Z_2^n \times Z_2^m \to Z_2^n$

this is simply a Boolean function.

So we achieve "cheap" computation of "forward" hash function $h$.

- But computation of pre-image of $h$ requires recovery of all "historic" inputs and states of finite state machine from its final state $y = q_l$. This requires search over space of all historic states $\Rightarrow$ very computationally costly.

# (Brief) Overview of Elliptic Curves for Public Key Cryptography

Observe that core concepts of RSA and Diffie-Hellman key exchange rest on a group $\mathbb{Z}_p^*$, p prime, and an associated generator g, with exponentiation using base g a low cost computation, while discrete logarithm is high cost computation (for sufficiently large p).

Feature required for public key cryptography is this one-way function (exponentiation above) with "costly" inverse (discrete logarithm).

In elliptic curve cryptography, the set of interest consists of pairs of points $(x, y) \in \mathbb{Z}_p \times \mathbb{Z}_p$, $p$ prime, $p \geq 5$. These pairs of points must satisfy a particular form of algebraic equation, modulo $p$.

Consider algebraic equation on

$(x, y) \in \mathbb{Z}_p \times \mathbb{Z}_p$

$$\mathrm{mod}(y^2, p) = \mathrm{mod}([x^3 + Ax + B], p) \quad (E)$$

with $A, B \in \mathbb{Z}_p$ <u>And</u>

$$\mathrm{mod}(4A^3 + 27B^2, p) \neq 0$$

{ this seemingly odd requirement ensures $x^3 + Ax + B$ has no repeated roots }

Let

$$\tilde{E}(\mathbb{Z}_p) \overset{\text{def}}{=} \left\{ (x, y) \in \mathbb{Z}_p \times \mathbb{Z}_p \;\middle|\; (E) \text{ is satisfied} \right\}$$

Let $\mathcal{O}$ denote an additional point "at infinity". Non-rigorously, think of $\mathcal{O} = (\underbrace{*}_{\text{wildcard}}, \infty)$, observe $\mathcal{O}$ clearly $\notin \mathbb{Z}_p \times \mathbb{Z}_p$

Let

$$E(\mathbb{Z}_p) = \tilde{E}(\mathbb{Z}_p) \cup \mathcal{O}$$

Example : Choose $p = 7$, $A = 3$, $B = 3$

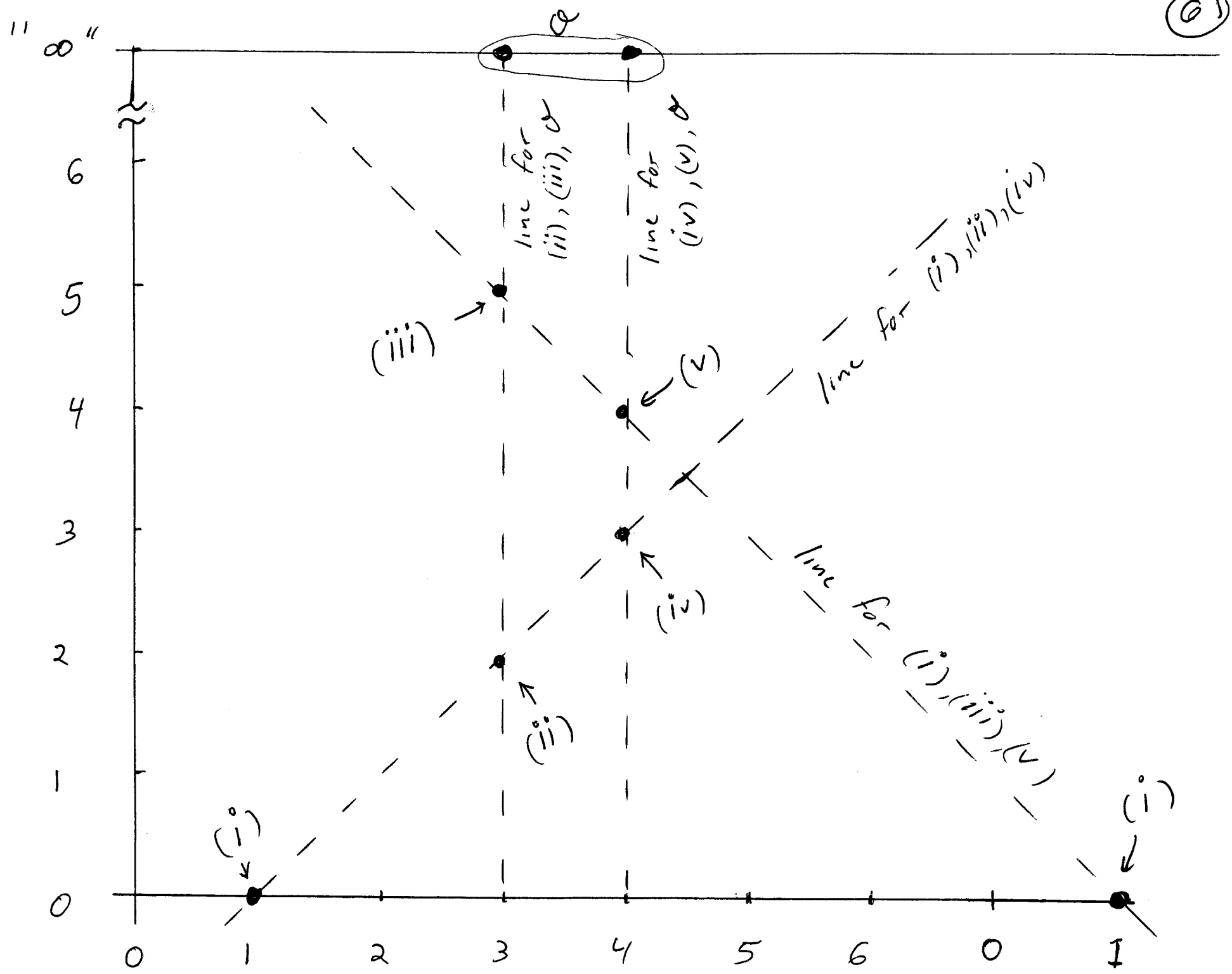
So $\tilde{E}(\mathbb{Z}_p)$ for this case are $(x, y)$ pairs satisfying

$$\mathrm{mod}(y^2, 7) = \mathrm{mod}\left(\left[x^3 + 3x + 3\right], 7\right)$$

Straightforward to confirm the following pairs lie in $\tilde{E}(\mathbb{Z}_7)$ (harder to confirm that these are <u>all</u> such points):

| | x | y |
|---|---|---|
| i) | 1 | 0 |
| ii) | 3 | 2 |
| iii) | 3 | 5 |
| iv) | 4 | 3 |
| v) | 4 | 4 |

} then to complete $E(\mathbb{Z}_2)$, add "a" point $\mathcal{O}$ with y at $\infty$, x arbitrary.


"∞"
∞
line for (iii), (iii), ∞
line for (iv), (v), ∞
line for (i), (ii), (iv)
line for (i), (iii), (v)
(iii)
(v)
(iv)
(ii)
(i)
(i)
6
5
4
3
2
1
0
0
1
2
3
4
5
6
0
1

FACT: Every line intersecting $E(\mathbb{Z}_p)$ intersects at exactly three points.

Consequence: We use this fact to define addition operation on $E(\mathbb{Z}_p)$:

- First, we simply define $\mathcal{O}$ as the additive identity:
  for any $P_1 \in E(\mathbb{Z}_p)$, $P_1 + \mathcal{O} = \mathcal{O} + P_1 = P_1$
- For any $P_1, P_2 \in E(\mathbb{Z}_p)$, $P_1 \neq P_2$, $P_3 = P_1 + P_2$ is that $P_3$ such that $(P_1, P_2, P_3)$ lie on line.

- For $P_1 = P_2 = P$, we use line tangent to curve at $P$ to find $P_3$ (admission: not fully illustrated in our $E(\mathbb{Z}_7)$ example)

Once addition is defined on our set (group) $E(\mathbb{Z}_p)$, we can define scaling by an integer $a \in \mathbb{Z}_p$:

i.e. for $P \in E(\mathbb{Z}_p)$, $a \in \mathbb{Z}_p$

$$a \cdot P = \underbrace{P + P + P \ldots + P}_{\text{"a" additions.}}$$

Q: How to adapt this to RSA Diffie-Hellman public key methods?

Ans: We need to identify our one-way function: what is the "easy" forward computation, what is the "hard" reverse computation?

- Given $a \in \mathbb{Z}_p$, $P_1 \in E(\mathbb{Z}_p)$ computing $P_2 = a \cdot P_1$ is the "cheap" forward computation.
- Given $P_1, P_2 \in E(\mathbb{Z}_p)$, computing $a \in \mathbb{Z}_p$ such that $P_2 = aP_1$ is hard.

- And, while details not pursued here, it's important that $a \cdot b \cdot P = b \cdot a \cdot P$ (analogy to exponentiation $g^{[ab]} = [g^a]^b = [g^b]^a$).

- What are the advantages?

  i) Evidence that the inverse problem above has even higher computational cost than the discrete logarithm,

  ii) This approach can achieve security with much smaller key space size (essentially, smaller $p$).

Other advanced topics of interest, in a nutshell ...

- Cryptography and quantum computing:

Quantum computer architectures are actually not that well suited to many traditional computations ... but, they are very well suited to the problem of factoring (into prime factors) very large integers. The difficulty of the factoring problem is at heart of difficulty of the discrete logarithm.

• Today's achievable size of quantum computers (number of "qubits") not yet a threat to RSA and related algorithms. Rough estimates ~ 10 million qubits capability in a quantum machine would begin to make "cracking" of present-day implementations of RSA feasible.

- Block-chain, and implementations in such applications as cryptocurrency. The underlying encryption decryption algorithms are exactly the class of public key approaches studied in 435. The novel feature employed by Bitcoin and others is implementation of algorithms in distributed architecture over the internet - concensus algorithms performing core computations