A SHA-3 Candidate

CRUNCH

Emmanuel Volte

27 february 2009

LEUVEN

CRUNCH TEAM

- Submitter: Jacques Patarin (PRISM, University of Versailles Saint Quentin, France)
- Louis Goubin (PRISM, UVSQ)
- Mickael Ivascot (PRISM, UVSQ)
- William Jalby (PRISM, UVSQ)
- Olivier Ly (LABRI, Bordeaux University, France)
- Valerie Nachef (University of Cergy-Pontoise, France)
- Joana Treger (UVSQ)
- Emmanuel Volte (UCP)

CRUNCH is a hash function based on the xor of two cipher blocks which are unbalanced Feistel Scheme with random expanding functions.

CRUNCH IS SIMPLE

CRUNCH IS SIMPLE

No special boxes

BUT

Random boxes (decimals of sinus)

CRUNCH IS SIMPLE

No special boxes

No tests

BUT

Random boxes (decimals of sinus)

Loops

CRUNCH IS SIMPLE

No special boxes

No tests

No addition, no multiplication, no shift . . .

BUT

Random boxes (decimals of sinus)

Loops

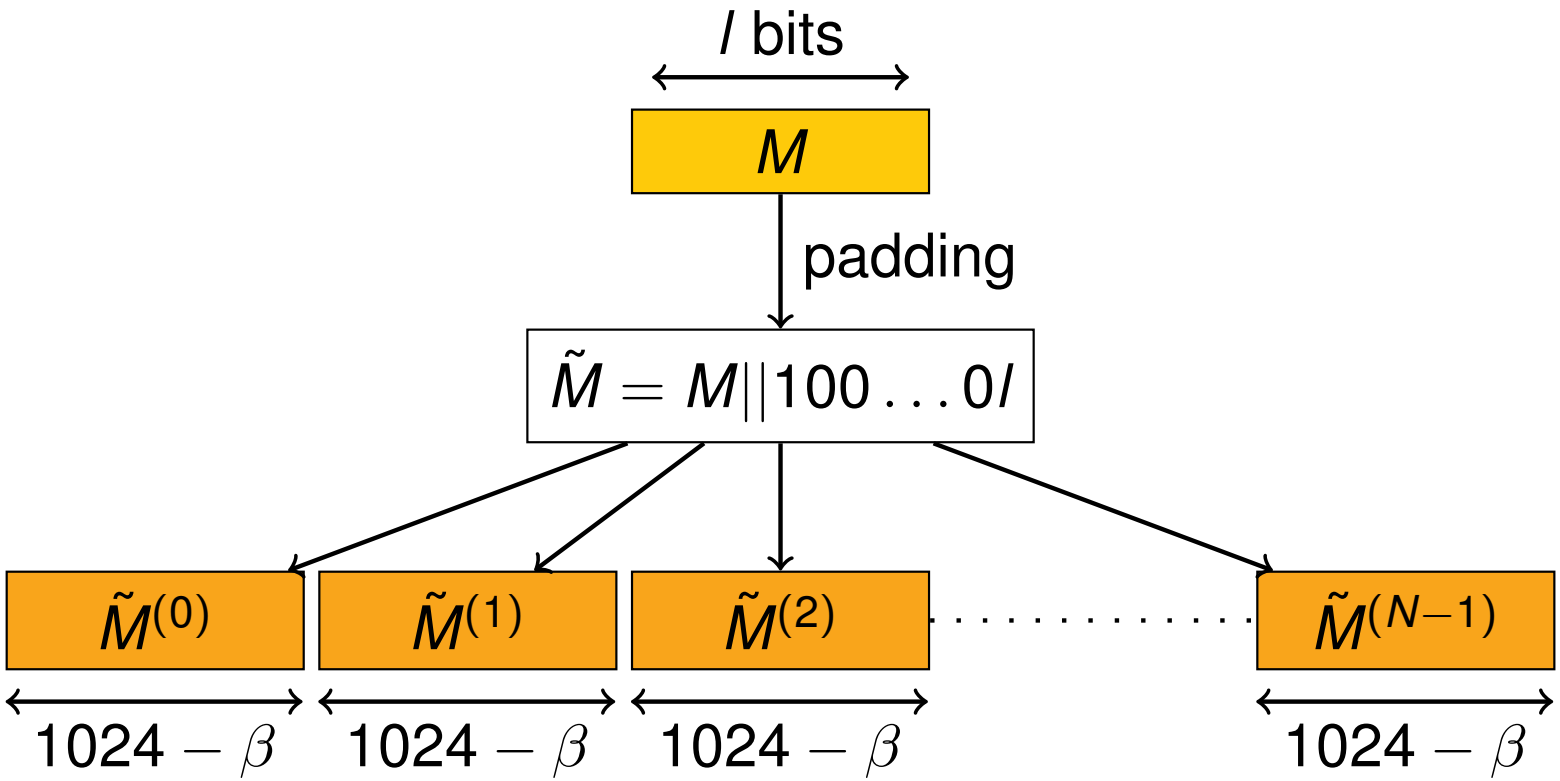
Xor

Complete code !

```
void Merkle_256(BlockType H[],BlockType mes[],int nb_merkle) {
  int n,k; int j_round,j,a,aa,aa2,p,block; int alpha;
  BlockType message[NE],res[NE]; BlockType i,i2;
  for(k=0;k<(NS);k++)  message[k]=H[k];
  for(n=0;n<nb_merkle;n++) {
    for(k=0;k<NS;k++)  res[k]=message[k];
    for(k=(NS) ;k<NE;k++) message[k]=mes[k-(NS)]; res[k]=message[k];
    mes=mes+(NE-NS); a=NE-(128/LONGSIZE); p=0;block=0; alpha=1;
    for(j_round=0;j_round<NBROUND;j_round++) {
      i2=(res[block]>>((NPM-1-p)*NBE))&(NA-1); i2=(i2*alpha)&0xff;
      i=(message[block]>>((NPM-1-p)*NBE))&(NA-1); i=(i*alpha)&0xff;
      aa2=(int)(a+NE*NA*16+i2*NE); aa=(int)(a+i*NE);
      for(j=0;j<NE;j++) {
         message[j]^=nbalea[aa]; aa++; res[j]^=nbalea[aa2]; aa2++;
      }
      if ((j_round & 15) == 15) { alpha=alpha+2;
        if ( (j_round&127) == 127) { a=NE-(128/LONGSIZE); }
        else { a-=15*NA*NE+128/LONGSIZE; }
      }
      else { a+=NA*NE; }
      p++; if (p==NPM) { block++; if (block==NE) block=0; p=0; }
    }
    for(j=SHIFT;j<SHIFT+NS;j++) res[j%NE]^=message[j%NE];
    for(j=0;j<NS;j++) message[j]=res[(j+SHIFT)%NE];
  }
  for(i=0;i<NS;i++) H[i]=message[i];
}
```

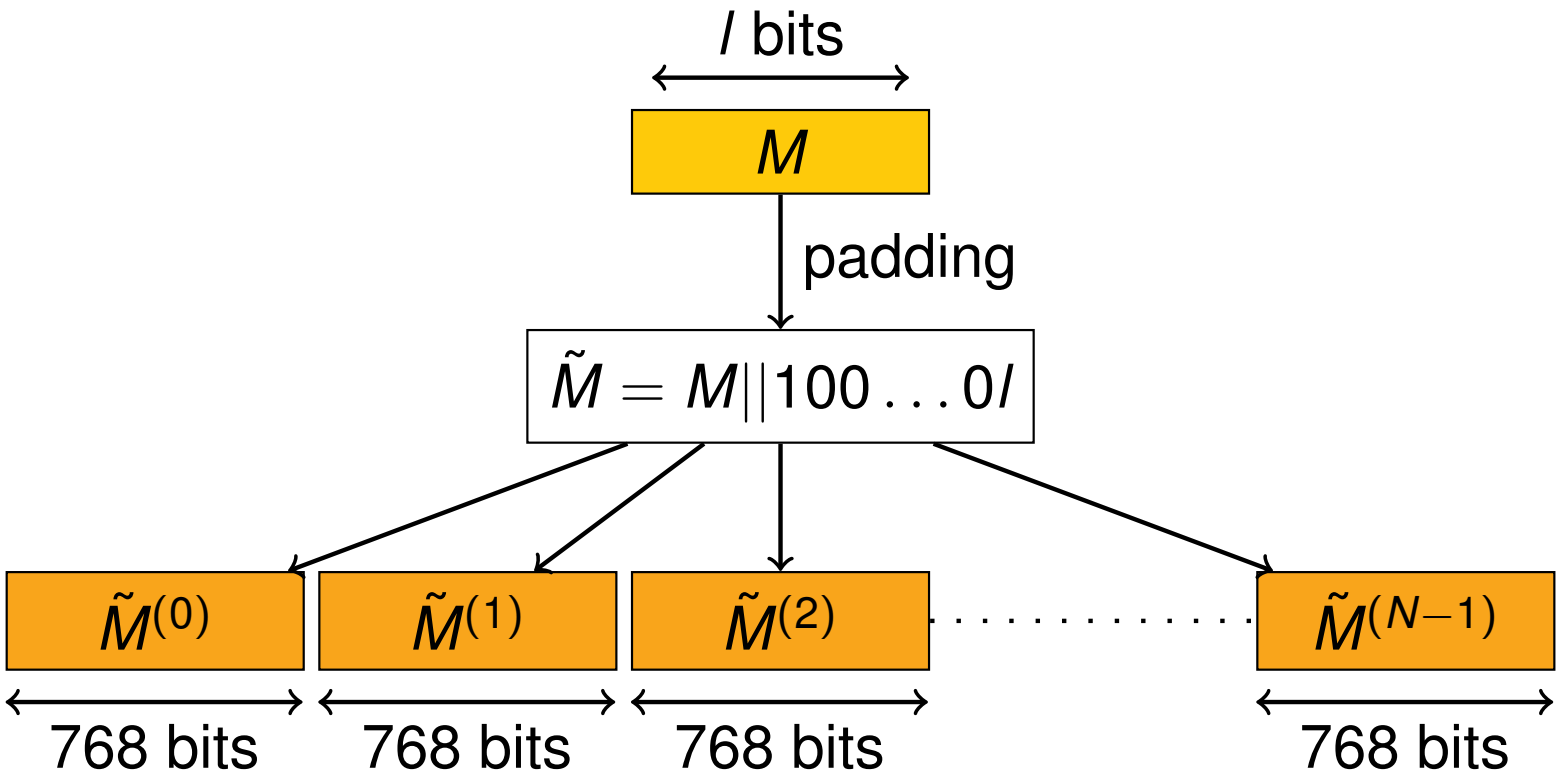
Initialisation

For CRUNCH-β :

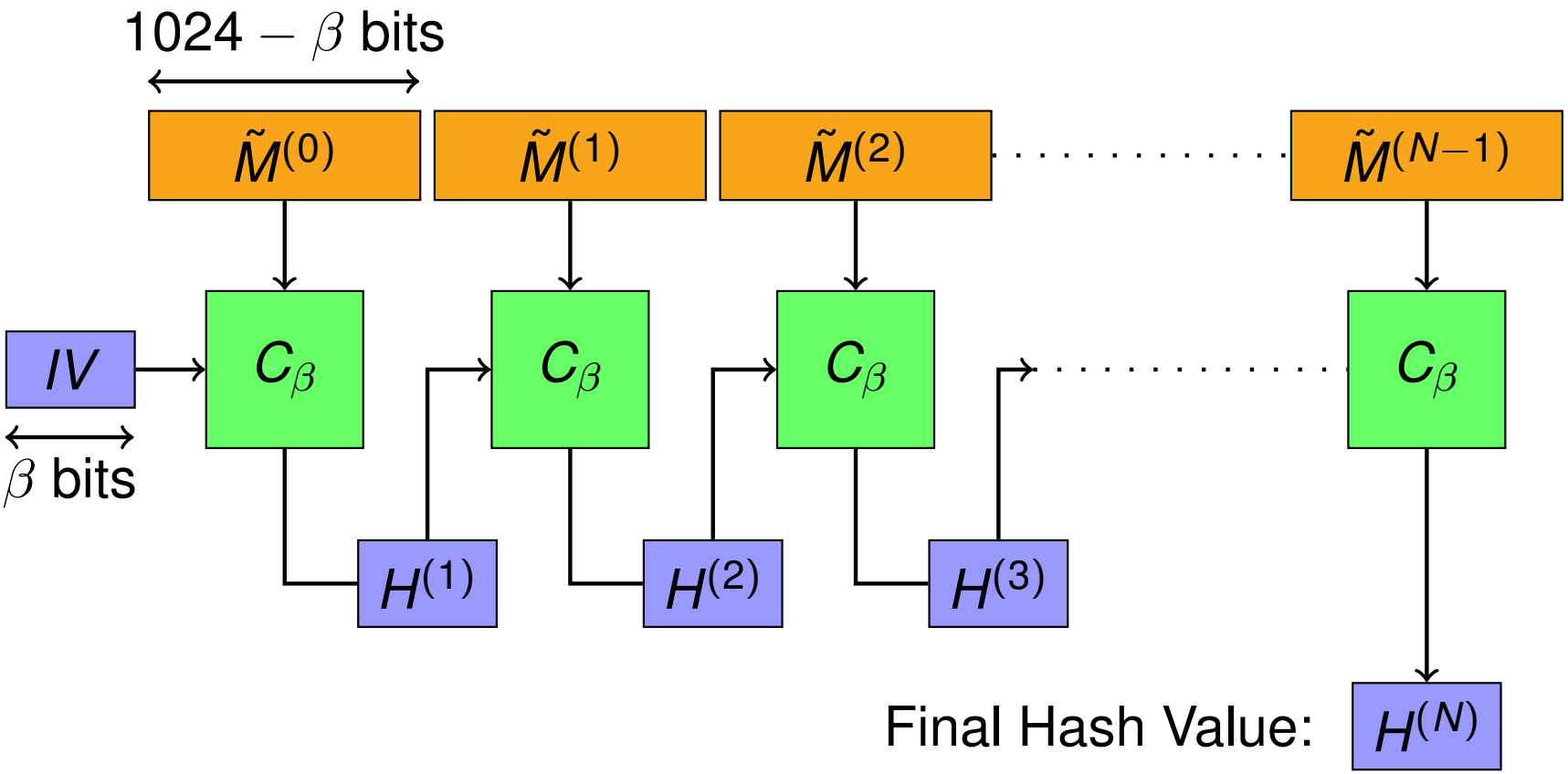


Initialisation

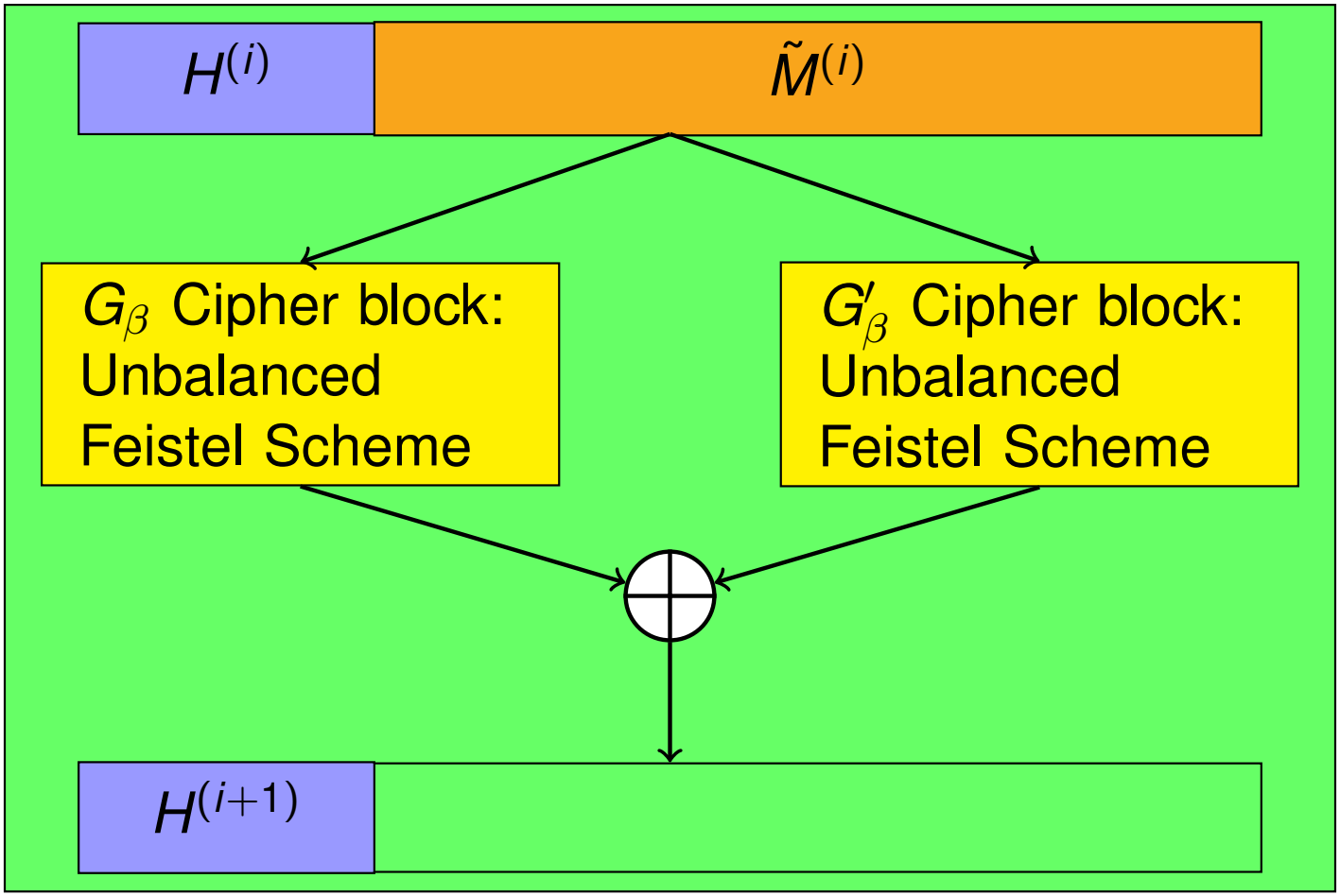
For CRUNCH-256 :



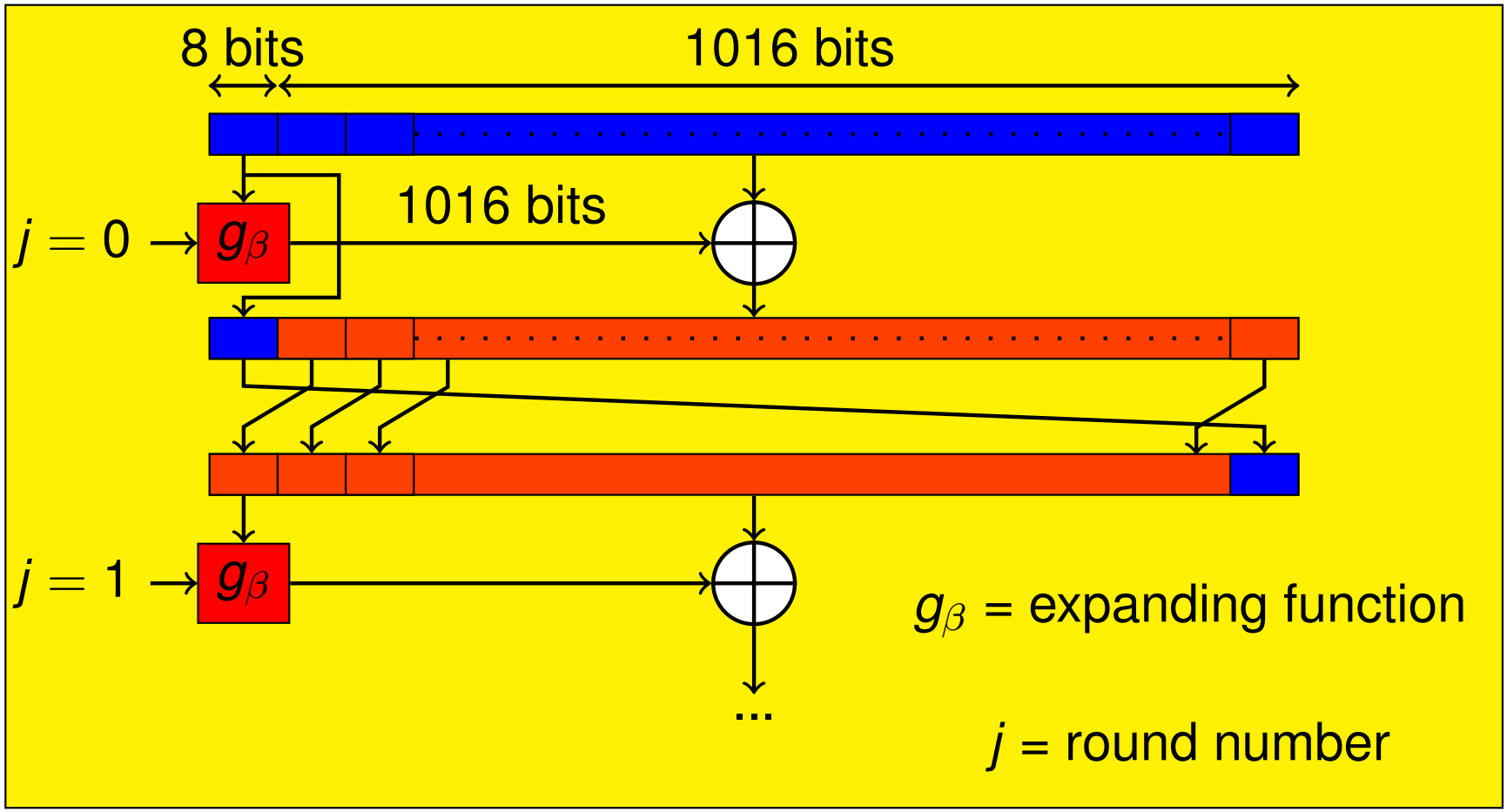
Chaining mode



Compression function



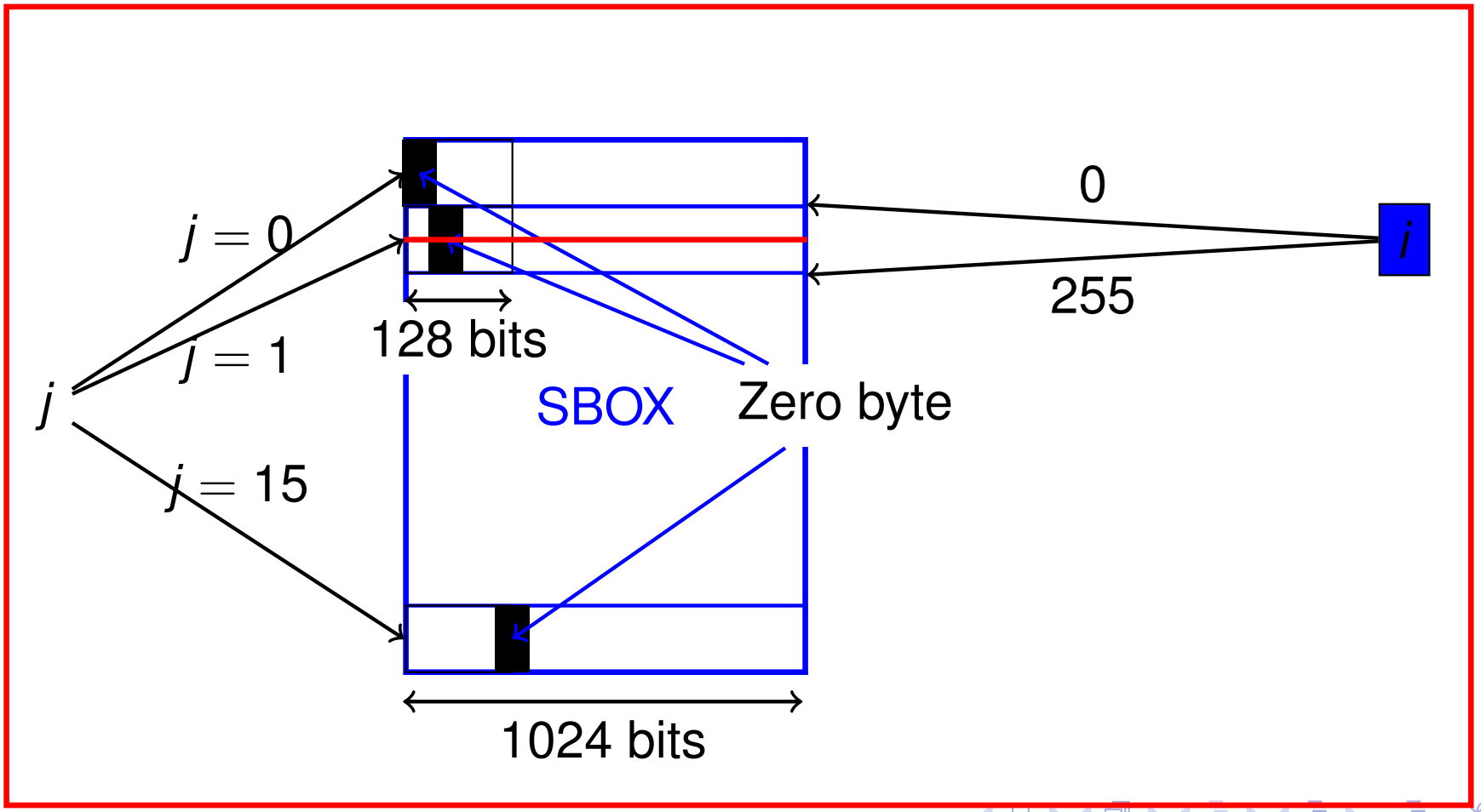
Unbalanced Feistel Scheme Cipher



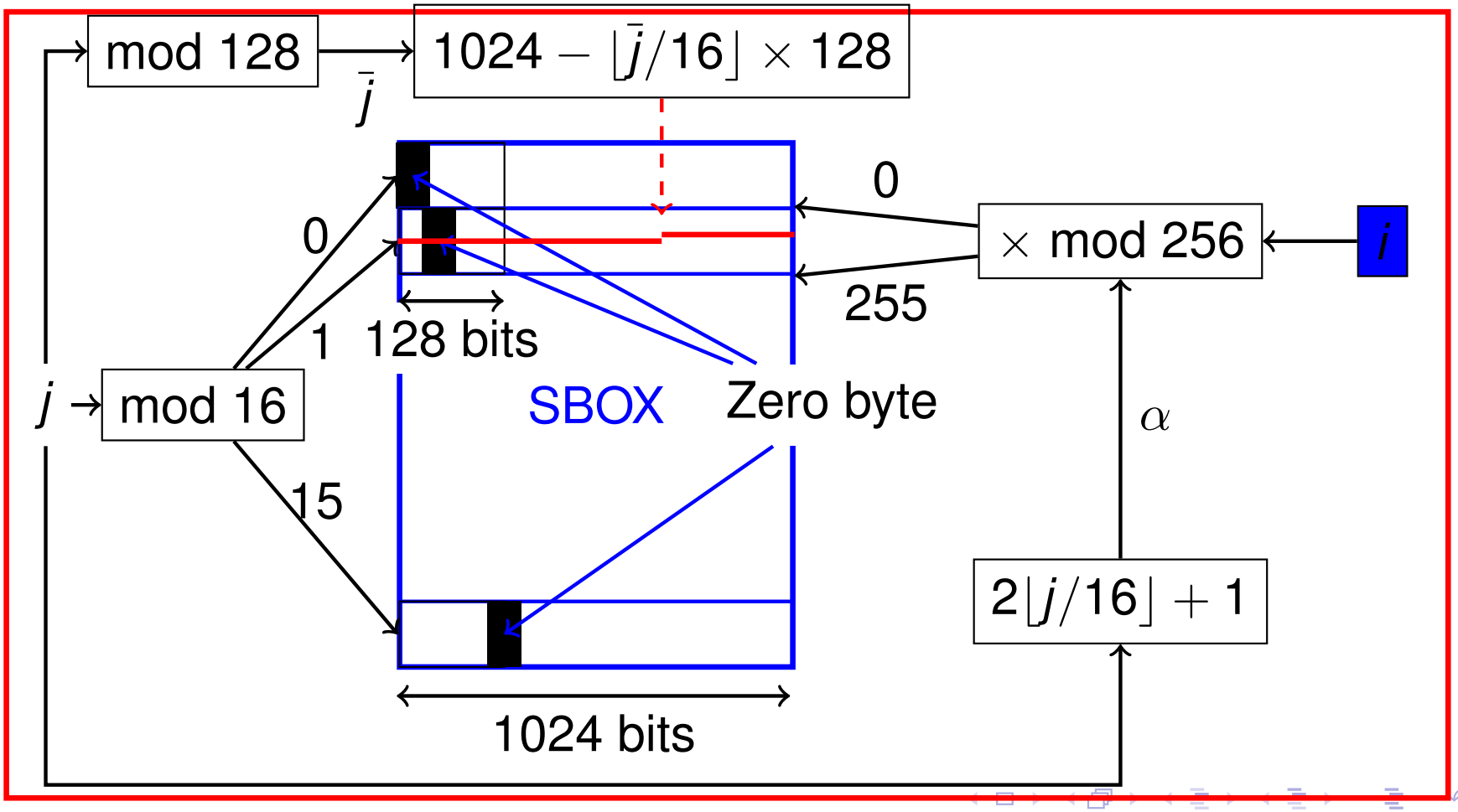
Number of rounds

Message digest (bits)	Security (bits)	Unbalanced Feistel Scheme F_d^{128} (Number of rounds)	Encryption Permutation (Number of rounds)
224	112	143	224
256	128	145	256
384	192	153	384
512	256	161	512

Expanding function (first 16 rounds)



Expanding function



Memory size/speed

Two S-boxes $\approx$ 1MB
So all can be in the L2 cache

For smart cards we can conpute S-Boxes on the fly.

CRUNCH speed rely on data access

CRUNCH do a lot of operations, each BYTE of the message is used at least 129 times for CRUNCH-256

64 bits processor

Processor: Intel Core 2 Duo E6400 @2.13GHz
(cache L1 data 32KB, cache L2 2MB)
RAM: 4GB DDR2 dual channel
compiler: icc v10.1, compilation options: -fast

Message digest (bits)	Message Size (MB)	Number of cycles	Speed (MB/s)
256	100	$169,5 * 10^8$	12,59
384	100	$296,2 * 10^8$	7,24
512	100	$469,7 * 10^8$	4,55

32-bit processor

Machine 1:
Processor: Intel Core 2 Duo E6400 @2.13GHz (cache L1 data 32KB,
RAM: 4GB DDR2 dual channel
OS: kubuntu 8.04.1 64bits with KDE 3.5
compiler: icc v10.1
compilation options: -O3 -march=pentium4

Message digest (bits)	Message Size (MB)	Number of cycles	Speed (MB/s)
224	100	$251,6 * 10^8$	8,48
256	100	$298,7 * 10^8$	7,15
384	100	$523,6 * 10^8$	4,08
512	100	$864,2 * 10^8$	2,47

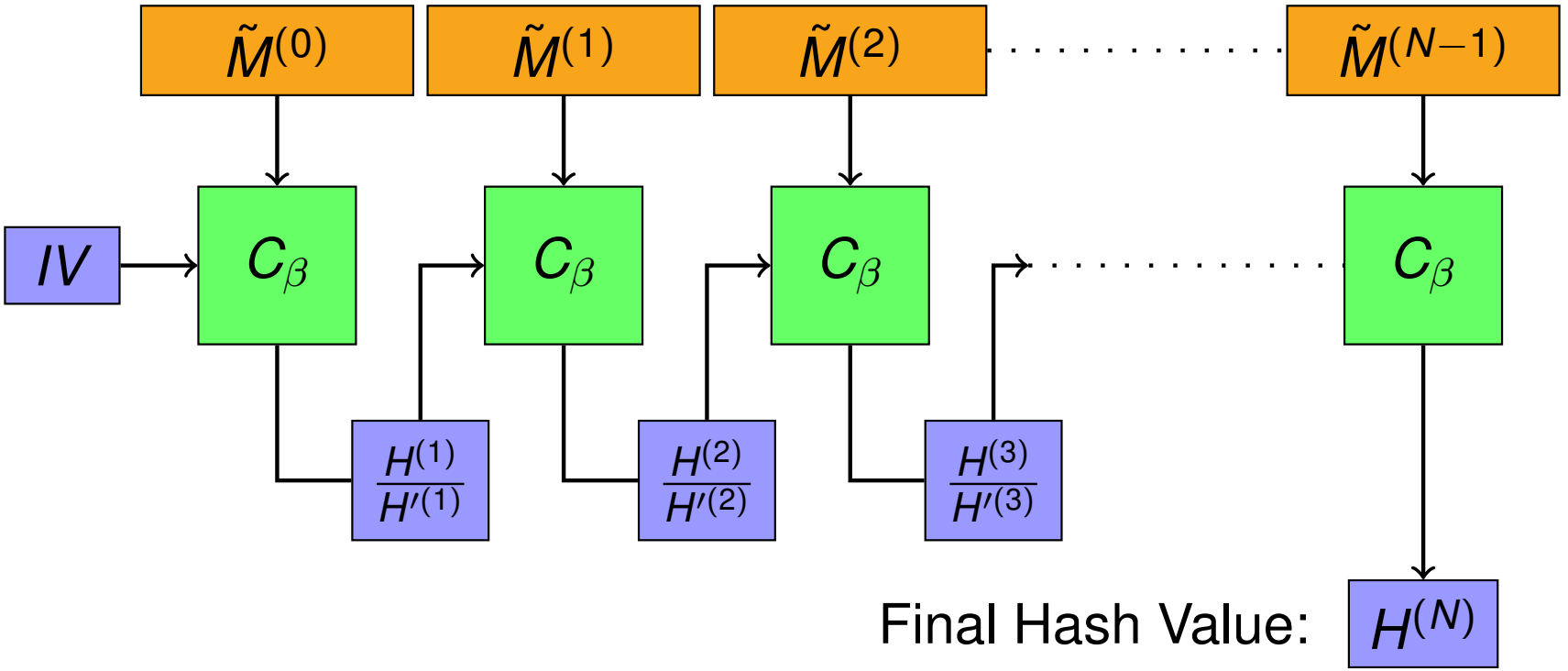
Definition

Definition

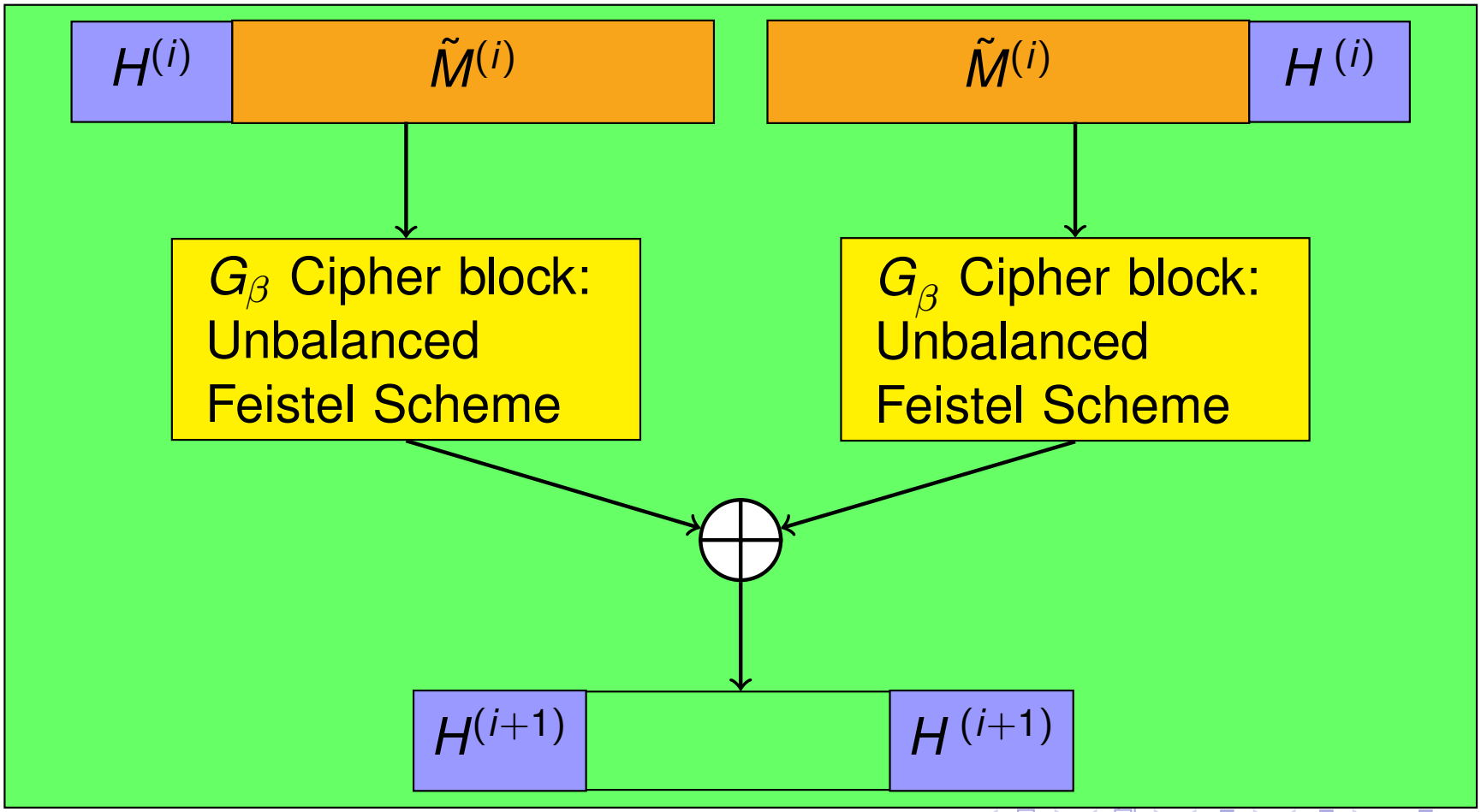
Let h be a hash function. We said that h is vulnerable to length extension attack if, for some x, we can guess $h(x||y)$ when we know $h(x)$ and y but not x.

It is obvious that crunch is vulnerable to such an attack but we can easily prevent it

A stronger chaining scheme (double pipe)



Compression function (small change)



Conclusion

Its simplicity is key to our design because it allows simple and efficient implementation on almost any microprocessor, it simplifies its protection and finally it makes easier to establish a direct relation between CRUNCH security and a generic (well known) security problem.