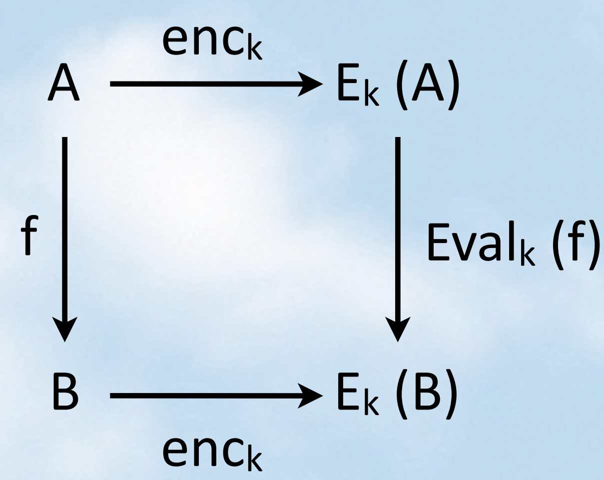
A Quick View of Homomorphic Encryption

John Launchbury

galois

Homomorphic Crypto

- A computation on the encrypted space produces the same result as the computation on the unencrypted space



galois

RSA
Semi-homomorphic

- RSA Settings
  - Size m, private key k, public key p
  - enc(x) = x^k mod m
- Let
  - Eval(*)(c,c') = cc' mod m
- Then

  enc(x)enc(y) mod m

  = (x^k mod m) (y^k mod m) mod m

  = $(xy)^k$ mod m

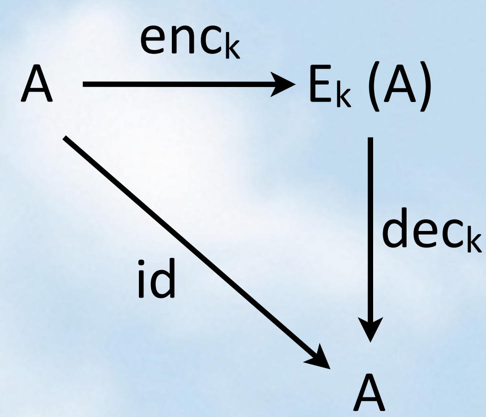
Homomorphic with respect to multiplication,

but not addition

galois

Homomorphic Crypto

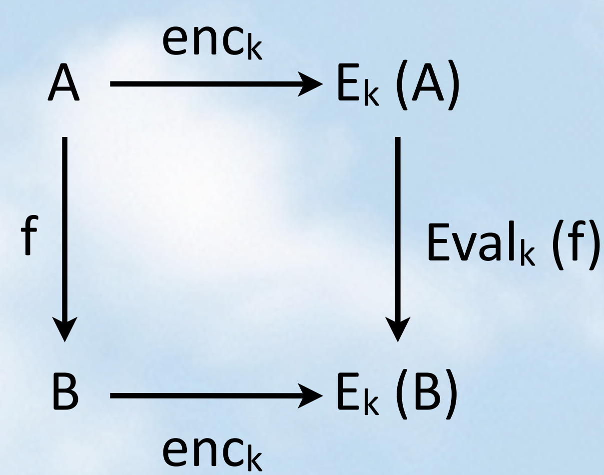
- Basic Setting
  - A, B are bit-vector types (i.e. finite products of Bool)
  - k is a cryptographic key
  - $E_k(A)$ is an integer type that may be *much* larger than A
- Encryption/Decryption
  - The encryption operation enc_k is typically a *random* multi-function
  - It has an inverse function dec_k



galois

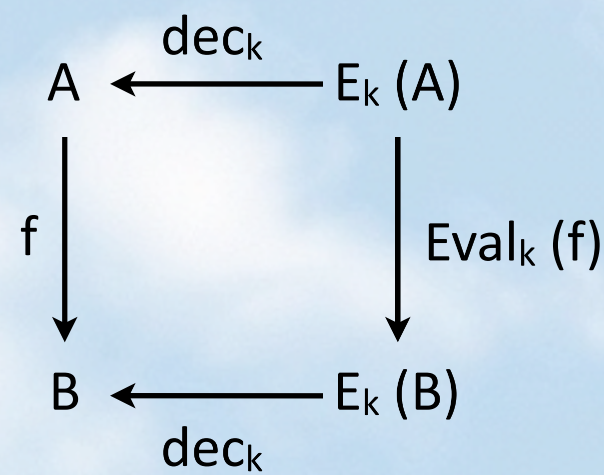
Homomorphic Crypto

- f is a function that can be represented as a boolean circuit
  - A boolean polynomial over AND, XOR
- $Eval_k(f)$ is a boolean circuit whose size is independent of the size of f
  - Compactness property



Homomorphic Crypto

- f is a function that can be represented as a boolean circuit
  - A boolean polynomial over AND, XOR
- $Eval_k(f)$ is a boolean circuit whose size is independent of the size of f
  - Compactness property



Simplified Gentry Scheme

- KeyGen : Bit-P
  - k <- random(P), odd
- Encrypt(m,k) : Key -> Bit-1 -> Integer
  - m' <- random(N), m' ≡ m(mod 2)
  - c <- m' + kq
- Decrypt(c,k) : Key -> Integer -> Bit-1
  - m <- (c mod k) mod 2

- Security settings
  - N = λ (e.g. 16)
  - P = λ^2 (e.g. 256)
  - Q = λ^5 (e.g. 1048576)
- q <- random(Q)

Simplified Gentry Scheme

- $m' \equiv m \pmod 2$ and $n' \equiv n \pmod 2$
- Addition

$$(m' + kq) + (n' + kq')$$
$$= (m'+n') + k(q+q')$$
$$= (m'+n') + kq''$$

- Multiplication

$$(m' + kq)(n' + kq')$$
$$= m'n' + k(m'q'+n'q+qq')$$
$$= m'n' + kq''$$

decrypt (c,k)
= (c mod k) mod 2

Simplified Gentry Scheme

- $m' \equiv m (\bmod 2)$ and $n' \equiv n (\bmod 2)$
- Addition

$$(m' + kq) + (n' + kq')$$
$$= (m'+n') + k(q+q')$$
$$= (m'+n') + kq''$$

- Multiplication

$$(m' + kq)(n' + kq')$$
$$= m'n' + k(m'q'+n'q+qq')$$
$$= m'n' + kq''$$

decrypt (c,k)
= (c mod k) mod 2

Noise

- Codes are “near multiples” of k with noise m
- Decrypt fails if the noise reaches P bits
  - A fresh encryption has N-bit noise
  - Adds add 1 bit to the noise
  - Mults double the noise

Homomorphic Crypto

- Encrypted decryption
  - Homomorphically lift decryption
  - Resets the noise
- Monads
  - This type is a multiplier operation
  - $E_k(A)$ is very much like a monad

$$E_k\,(E_k\,(A))$$

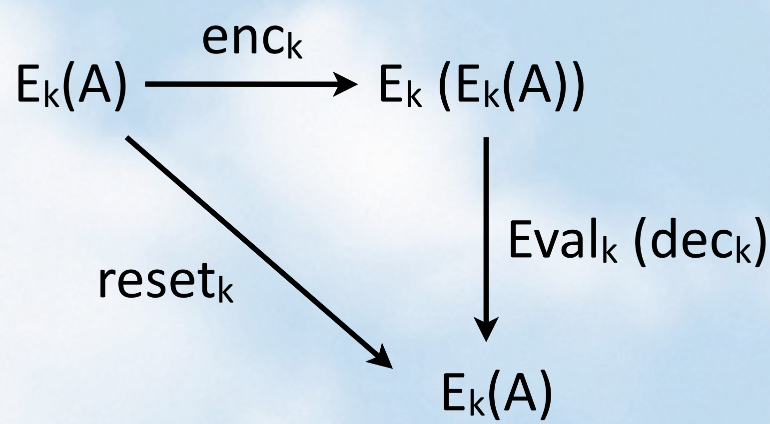
$$\downarrow \; Eval_k\,(dec_k\,)$$

$$E_k\,(A)$$

galois

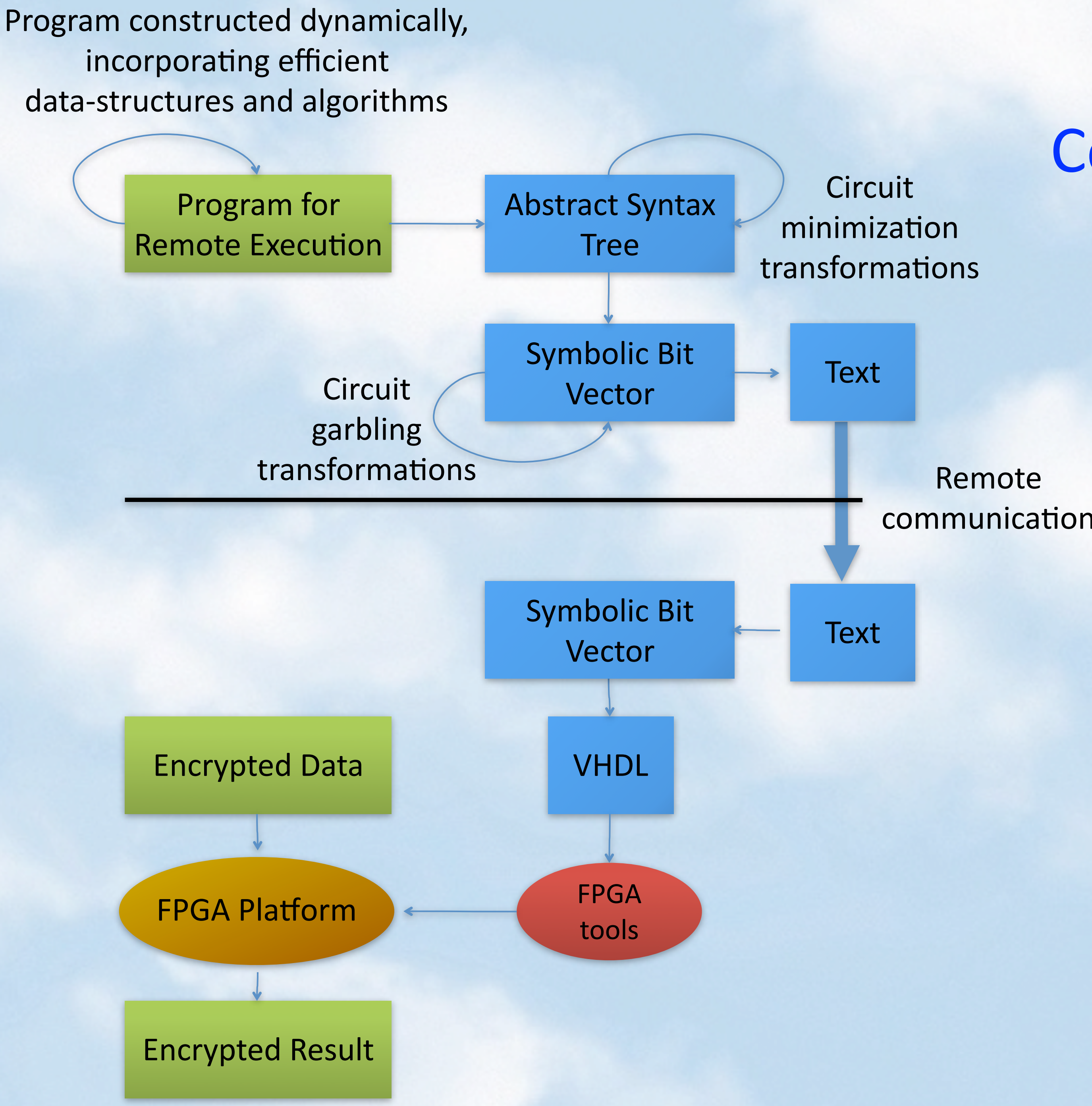
Noise Reduction

- May need a special formulation of dec_k to make it small enough
- Low degree polynomial



galois

Phases of Computation



galois

Proxy Crypto

- Operations on different keys can be combined
  - Proxy cryptography
  - Translate from one key to another, without ever producing plaintext in the process