Cryptography & Statistics: a short introduction

Louis J. M. Aslett (louis.aslett@durham.ac.uk)

Department of Mathematical Sciences
Durham University

Short Course, Part I

54th Gregynog Statistical Conference
March 2018

Outline

1. Standard Encryption
   - Discussion of encryption concepts to set the scene.
2. Homomorphic Encryption
   - Definition and high level discussion of homomorphic schemes.
3. Fan & Vercauteren (2012)
   - In depth look at this specific homomorphic encryption scheme.
   - Some further discussion on polynomial Chinese remainder Theorem.
4. Software
   - Discussion of implementation issues and `HomomorphicEncryption` R package.
5. Multiparty Computing
   - Homomorphic secret sharing with information theoretic security levels.

Standard Encryption

Encryption basics (I)

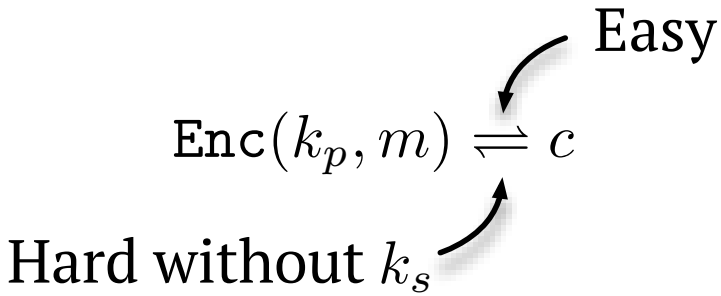
Broadly speaking, an encryption scheme consists of:

- Unencrypted object, $m \in M$, referred to as a *message*.
  - M is the *message space*.
- Encrypted version, $c \in C$, referred to as a *cipher text*.
  - C is the *cipher text space*.
- Single $(k_s) \in K_s$, or pair $(k_s, k_p) \in K_s \times K_p$, of 'keys'.
  - Single key means secret key scheme;
  - Pair of keys means public key scheme.
- Injective map, $\texttt{Enc} : K_p \times M \to C$.
  - not necessarily a function, message can encrypt to different cipher texts.
- Surjective function, $\texttt{Dec} : K_s \times C \to M$.
- Enc and Dec satisfy:

$$m = \texttt{Dec}(k_s, \texttt{Enc}(k_p, m)) \quad \forall\, m \in M$$

Encryption basics (II)

Fundamental point is ...



$$\texttt{Dec}(k_s, c) = m$$

The *security level* of an encryption scheme is the order of the number of operations required to crack it (decrypt without k_s).

Clearly, an upper bound on the security of an encryption scheme is $O(|K_s|)$, since a brute force attack which tries every possible secret key will succeed.

Concepts: Public key -vs- private key

Presumably public key schemes are always better: can just choose not to distribute k_p?

Not really. Public key schemes tend to:

- have much larger cipher texts than messages, so are space inefficient.
- have greater computational cost, so are compute inefficient.
- rely on complex mathematical constructions rather than bit-level operations, so are hard to design custom hardware for.

Hence, private key schemes still involved in almost all cryptography, perhaps wrapped in a public key scheme. More anon ...

Some common schemes (history, I)

- DES or Triple-DES. Secret-key scheme with 56-bit keys.
  - DES: block cipher algorithm ... bit fiddling transformations which incorporate key.
  - TDEA: $\mathtt{Enc}(., m) := \mathtt{Enc}(k_{s3}, \mathtt{Dec}(k_{s2}, \mathtt{Enc}(k_{s1}, m)))$.
- RSA. Famously the first practical public-key scheme, based on prime number pairs.
  - $k_p = (n, e)$ where:
    - $n = pq$ for p, q prime;
    - e integer, $1 < e < \phi(n)$, $\gcd(e, \phi(n)) = 1$
    - $\mathtt{Enc}(k_p, m) := m^e \mod n$
  - $k_s = (d)$ where $d = e^{-1} \mod \phi(n)$
  - $\mathtt{Dec}(k_s, c) := c^d \mod n$

Some common schemes (history, I)

- DES or Triple-DES. Secret-key scheme with 56-bit keys.
  - DES: block cipher algorithm ... bit fiddling transformations which incorporate key.
  - TDEA: $\texttt{Enc}(., m) := \texttt{Enc}(k_{s3}, \texttt{Dec}(k_{s2}, \texttt{Enc}(k_{s1}, m)))$.
- RSA. Famously the first practical public-key scheme, based on prime number pairs.
  - $k_p = (n, e)$ where:
    - $n = pq$ for p, q prime;
    - e integer, $1 < e < \phi(n)$, $\gcd(e, \phi(n)) = 1$
    - $\texttt{Enc}(k_p, m) := m^e \mod n$
  - $k_s = (d)$ where $d = e^{-1} \mod \phi(n)$
  - $\texttt{Dec}(k_s, c) := c^d \mod n$

Demo: 000_RSA.R

Some common schemes (history, II)

PGP. Arguably first encryption software popular with regular users.

- Uses RSA to encrypt a Triple-DES key, k'_s
- Uses Triple-DES to encrypt a compressed version of message

$$\mathsf{Enc}(k_p, m) = (\mathsf{Enc}_{RSA}(k_p, k'_s), \mathsf{Enc}_{TDEA}(k'_s, m))$$

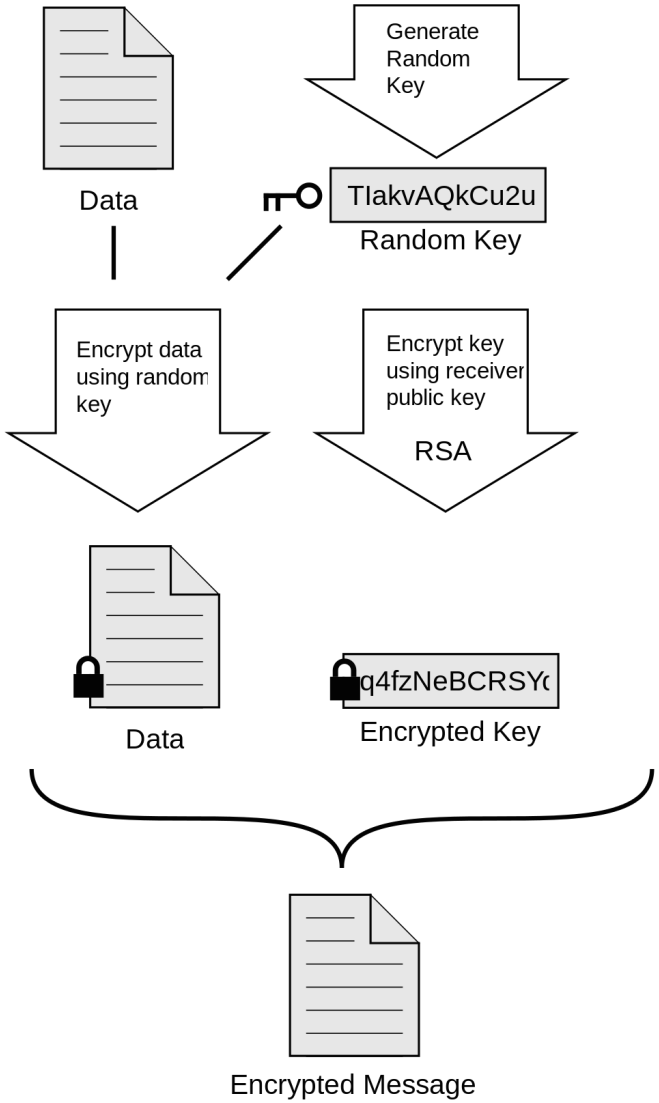


Image by xaedes & jfreax & Acdx [CC BY-SA 3.0]

Some common schemes (today)

- AES (Advanced Encryption Standard). Secret-key scheme which has superceded DES and Triple-DES. Now an industry standard.
  - Use wifi with WPA2? All traffic encrypted with AES unless you use TKIP for backwards compatability.
  - Own an iPhone/iPad? The internal flash storage is automatically encrypted using 256-bit AES.
  - Most Intel CPUs since 2010 include hardware AES acceleration.
  - Required for US federal encryption since 2014.
  - Brute force attacks on AES-128 require 2 billion years running 1 trillion machines capable of testing 1 billion keys a second.
- TLS/SSL. Every time you visit a secure website.
  - RSA typically still used to verify identity and exchange secret key.
  - Triple-DES or AES used to encrypt the webpage content.

Problem: ‘Brittle’ encryption

Most cryptography schemes are ‘brittle’ in that we can’t manipulate the message contained in the mathematical vault: must decrypt to compute, then encrypt the result. i.e. seems only useful for shipping round static data!

Problem: 'Brittle' encryption

Most cryptography schemes are 'brittle' in that we can't manipulate the message contained in the mathematical vault: must decrypt to compute, then encrypt the result. i.e. seems only useful for shipping round static data!

Continue demo: 000_RSA.R

Problem: 'Brittle' encryption

Most cryptography schemes are 'brittle' in that we can't manipulate the message contained in the mathematical vault: must decrypt to compute, then encrypt the result. i.e. seems only useful for shipping round static data!

Continue demo: 000_RSA.R

In other words, if

$$c_1 := \mathsf{Enc}(k_p, m_1)$$
$$c_2 := \mathsf{Enc}(k_p, m_2)$$

then in general, for a given function $g(\cdot,\cdot)$, $\nexists\, f(\cdot,\cdot)$ (not requiring k_s) such that

$$\mathsf{Dec}(k_s, f(c_1, c_2)) = g(m_1, m_2) \quad \forall\, m_1, m_2 \in M$$

Amazingly, for naïve RSA $f(x,y) \equiv g(x,y) := x \times y$ works.

Still problems: RSA

RSA has a serious problem though: it is a completely deterministic encryption scheme.

$\Longrightarrow$ easy to do 'dictionary' attacks if you know plausible data.

Not such a problem for e.g. PGP as Triple-DES secret key is random (though security now reduced to length of Triple-DES key).

BIG problem for statisticians! (e.g. binary classification)

Still problems: RSA

RSA has a serious problem though: it is a completely deterministic encryption scheme.

$\Longrightarrow$ easy to do 'dictionary' attacks if you know plausible data.

Not such a problem for e.g. PGP as Triple-DES secret key is random (though security now reduced to length of Triple-DES key).

BIG problem for statisticians! (e.g. binary classification)

Finish demo: 000_RSA.R

Still problems: RSA

RSA has a serious problem though: it is a completely deterministic encryption scheme.

$\Longrightarrow$ easy to do 'dictionary' attacks if you know plausible data.

Not such a problem for e.g. PGP as Triple-DES secret key is random (though security now reduced to length of Triple-DES key).

BIG problem for statisticians! (e.g. binary classification)

Finish demo: 000_RSA.R

Fix: Real-world RSA implementations pad the message with noise, but this breaks the nice computing property!

Concept: Semantic security

Definition (Semantic security)

An encryption scheme is said to be *semantically secure* if knowledge of the cipher text for some message has vanishingly small probability of revealing further information about any other encrypted message.

Informally: repeated encryption of same message renders different and seemingly unrelated cipher texts with high probability.

Why do we care? For private key scheme you don't. However, in a public key scheme where $|M| \ll |K_s|$ or probable messages are known, an attacker can perform a 'chosen plaintext attack' if not semantically secure — simply encrypt using the public key and compare.

Homomorphic Encryption

Homomorphic Encryption

Rivest et al. (1978) hypothesised that a limited set of functions may be possible to compute encrypted: specifically those involving addition and multiplication.

Definition (Homomorphic encryption scheme)

An encryption scheme is said to be *homomorphic* if there is a set of operations $\circ \in \mathcal{F}_M$ acting in message space (such as addition) that have corresponding operations $\diamond \in \mathcal{F}_C$ acting in cipher text space satisfying the property:

$$\mathtt{Dec}(k_s, \mathtt{Enc}(k_p, m_1) \diamond \mathtt{Enc}(k_p, m_2)) = m_1 \circ m_2 \quad \forall\, m_1, m_2 \in M$$

A scheme is *fully homomorphic* if $\mathcal{F}_M = \{+, \times\}$ and an arbitrary number of such operations are possible.

The first fully homomorphic scheme was not found until Gentry (2009)

Homomorphic Encryption

Definition (Homomorphic encryption scheme)

An encryption scheme is said to be *homomorphic* if there is a set of operations $\circ \in \mathcal{F}_M$ acting in message space (such as addition) that have corresponding operations $\diamond \in \mathcal{F}_C$ acting in cipher text space satisfying the property:

$$\mathsf{Dec}(k_s, \mathsf{Enc}(k_p, m_1) \diamond \mathsf{Enc}(k_p, m_2)) = m_1 \circ m_2 \quad \forall\, m_1, m_2 \in M$$

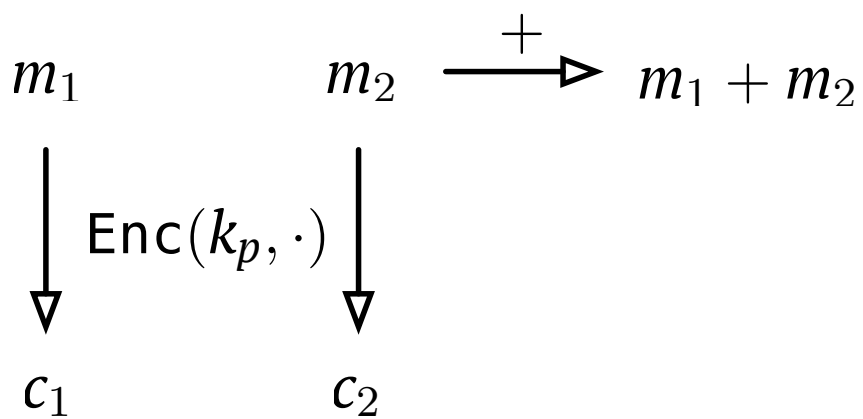
$$m_1 \qquad m_2 \xrightarrow{+} m_1 + m_2$$

Homomorphic Encryption

Definition (Homomorphic encryption scheme)

An encryption scheme is said to be *homomorphic* if there is a set of operations $\circ \in \mathcal{F}_M$ acting in message space (such as addition) that have corresponding operations $\diamond \in \mathcal{F}_C$ acting in cipher text space satisfying the property:

$$\mathsf{Dec}(k_s, \mathsf{Enc}(k_p, m_1) \diamond \mathsf{Enc}(k_p, m_2)) = m_1 \circ m_2 \quad \forall\, m_1, m_2 \in M$$

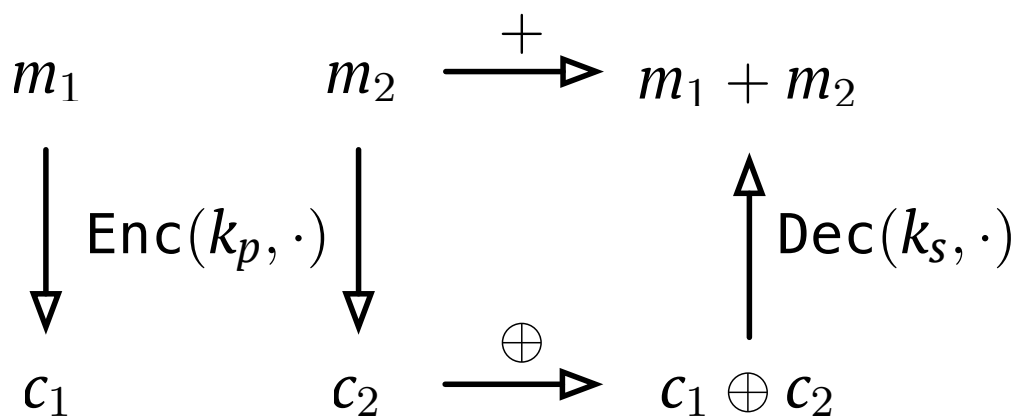


Homomorphic Encryption

Definition (Homomorphic encryption scheme)

An encryption scheme is said to be *homomorphic* if there is a set of operations $\circ \in \mathcal{F}_M$ acting in message space (such as addition) that have corresponding operations $\diamond \in \mathcal{F}_C$ acting in cipher text space satisfying the property:

$$\mathsf{Dec}(k_s, \mathsf{Enc}(k_p, m_1) \diamond \mathsf{Enc}(k_p, m_2)) = m_1 \circ m_2 \quad \forall\, m_1, m_2 \in M$$

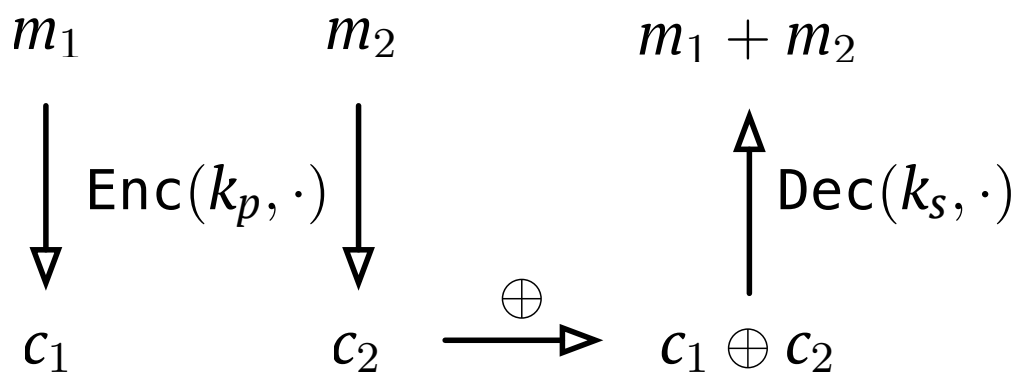


Homomorphic Encryption

Definition (Homomorphic encryption scheme)

An encryption scheme is said to be *homomorphic* if there is a set of operations $\circ \in \mathcal{F}_M$ acting in message space (such as addition) that have corresponding operations $\diamond \in \mathcal{F}_C$ acting in cipher text space satisfying the property:

$$\mathsf{Dec}(k_s, \mathsf{Enc}(k_p, m_1) \diamond \mathsf{Enc}(k_p, m_2)) = m_1 \circ m_2 \quad \forall\, m_1, m_2 \in M$$



Homomorphic + Semantic Security $\neq$ Homomorphism

Note, the name is inspired by the definition of a homomorphism between groups.

Definition (Group homomorphism)

Given two groups $(G, \circ)$ and $(H, \diamond)$, a *group homomorphism* from $(G, \circ)$ to $(H, \diamond)$ is a function $f : G \to H$ such that $\forall\, u, v \in G$,

$$f(u \circ v) = f(u) \diamond f(v)$$

But, if $f()$ is encryption, then homomorphic encryption is usually *not* a group homomorphism due to semantic security!

i.e.

$$f^{-1}(f(u \circ v)) = f^{-1}(f(u) \diamond f(v))$$

RSA as a homomorphic scheme

We can see now why RSA had the multiplicative homomorphic property.

$$\mathcal{F}_M = \{\times\}, \mathcal{F}_C = \{\times\}$$

$$\begin{aligned}
\mathsf{Enc}(k_p, m_1) \times \mathsf{Enc}(k_p, m_2) &= (m_1^e \mod n) \times (m_2^e \mod n) \\
&= (m_1 m_2)^e \mod n \\
&= \mathsf{Enc}(k_p, m_1 m_2)
\end{aligned}$$

Note that the final equality indicates a lack of semantic security, so actually RSA is not great when we want to encrypt plain old integer data as it will be very vulnerable to chosen plaintext attack.

Why $+$ and $\times$?

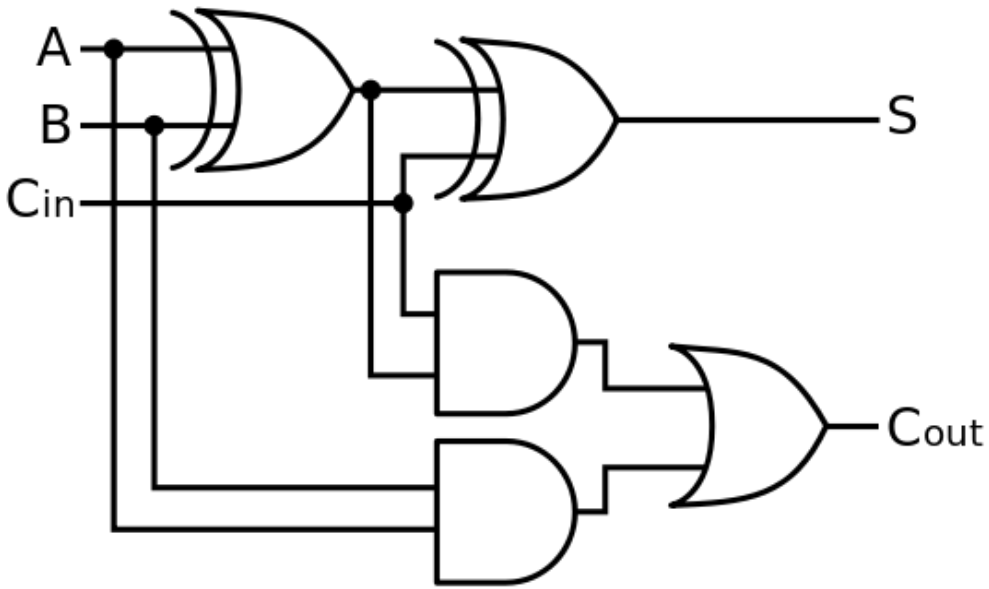
Addition and multiplication seem pretty limiting, why all the excitement if this is all that is possible?

Note that if $M = \mathrm{GF}(2)$, then:

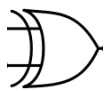
- $+ \equiv \veebar$, i.e. XOR, 'exclusive or'
- $\times \equiv \wedge$, i.e. AND, 'and'

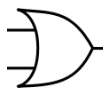
Moreover, *any* electronic logic gate can be constructed using only XOR and AND gates. Therefore, theoretically any operation on a computer can be performed encrypted.

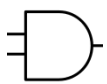
Adding n-bit integers encoded in GF$(2)^n$ (I)



A, B input bits
C_{in} carry bit
S result bit
C_{out} carry out bit

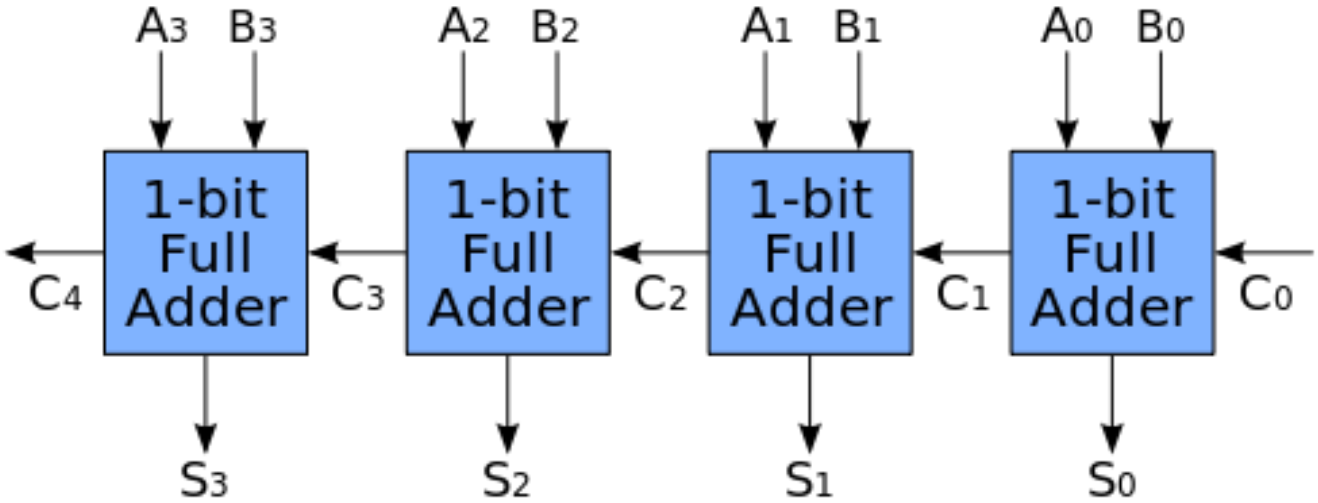
 $\equiv a + b$

 $\equiv a + b + ab$

 $\equiv ab$

$$S = A + B + C_{\text{in}}$$
$$C_{\text{out}} = (A + B)C_{\text{in}} + AB + (A + B)ABC_{\text{in}}$$

Adding n-bit integers encoded in GF$(2)^n$ (I)



$$S_0 = A_0 + B_0$$
$$S_1 = A_0B_0 + A_1 + B_1$$
$$S_2 = (A_1 + B_1)A_0A_1B_0B_1 + (A_1 + B_1)A_0B_0 + A_1B_1 + A_2 + B_2$$
$$S_3 = \big((A_1 + B_1)A_0A_1B_0B_1 + (A_1 + B_1)A_0B_0 + A_1B_1\big)(A_2 + B_2)A_2B_2$$
$$+ \big((A_1 + B_1)A_0A_1B_0B_1 + (A_1 + B_1)A_0B_0 + A_1B_1\big)(A_2 + B_2)$$
$$+ A_2B_2 + A_3 + B_3$$

This is just $n = 4$ (i.e. max value 15) ... gets messy quickly!

Limitations of homomorphic encryption

1. Message space
   - Commonly only easy to encrypt binary/integers
2. Cipher text size
   - Present schemes all inflate the size of data substantially (e.g. 1MB $\rightarrow$ 16.4GB)
3. Computational cost
   - 1000's additions per sec
   - $\approx$ 50 multiplications per sec
4. Division and comparison operations
   - Impossible!
5. Depth of operations
   - After a certain depth of multiplications, need to 'refresh' cipher text: hugely time consuming, so avoid!

‘Depth’ of operations

Fully homomorphic encryption schemes only support $+$ and $\times$, hence all function evaluations are simply multivariate polynomials:

$$\sum_{j=1}^{p} \prod_{i=1}^{m} x_i^{n_{ij}}$$

We define the *depth* of multiplication operations required to be:

$$\max \left\{ \sum_{i=1}^{m} n_{ij} : j \in \{1, \ldots, p\} \right\} - 1$$

'Depth' of operations

Fully homomorphic encryption schemes only support $+$ and $\times$, hence all function evaluations are simply multivariate polynomials:

$$\sum_{j=1}^{p} \prod_{i=1}^{m} x_i^{n_{ij}}$$

We define the *depth* of multiplication operations required to be:

$$\max \left\{ \sum_{i=1}^{m} n_{ij} : j \in \{1, \ldots, p\} \right\} - 1$$

$$\begin{aligned} S_2 &= (A_1 + B_1)A_0A_1B_0B_1 + (A_1 + B_1)A_0B_0 + A_1B_1 + A_2 + B_2 \\ &= A_0A_1^2B_0B_1 + A_0A_1B_0B_1^2 + A_0B_0A_1 + A_0B_0B_1 \\ &\quad + A_1B_1 + A_2 + B_2 \end{aligned}$$

has depth 4.

We will see why the depth matters in next section.

‘Bootstrap’ — cipher text refreshing

Unrelated to statistical term ‘bootstrap’.

See in next section, operations with cipher texts in a semantically secure scheme have a noise component which increases (potentially geometrically) with multiplication operations. Hence, if the function evaluated exceeds a certain depth then noise will overwhelm the message.

The breakthrough by Gentry (2009) was constructing a decryption algorithm simple enough to *run itself encrypted*.

Essentially, if you can do (v loosely speaking):

$$c' = \mathtt{Dec}(\mathtt{Enc}(k_p, k_s), c)$$

then c' will be a cipher text representing the same message as c, but with noise level reset to that of a fresh cipher text.

Fan & Vercauteren (2012)

Fan & Vercauteren (2012) scheme : notation

- $\mathbb{Z}_q = \{n : n \in \mathbb{Z}, -q/2 < n \leq q/2\}$
- $[a]_q$ is unique integer in $\mathbb{Z}_q$ st $[a]_q = a \mod q$
- $\mathbb{Z}[x], \mathbb{Z}_q[x]$ denote polynomials with coefficients in $\mathbb{Z}$ and $\mathbb{Z}_q$ respectively

Fan & Vercauteren (2012) scheme : notation

- $\mathbb{Z}_q = \{n : n \in \mathbb{Z}, -q/2 < n \leq q/2\}$
- $[a]_q$ is unique integer in $\mathbb{Z}_q$ st $[a]_q = a \mod q$
- $\mathbb{Z}[x], \mathbb{Z}_q[x]$ denote polynomials with coefficients in $\mathbb{Z}$ and $\mathbb{Z}_q$ respectively
- $\Phi_n(x)$ is nth cyclotomic polynomial
- $\Phi_{2^d}(x) = x^{2^{d-1}} + 1$

Cyclotomic polynomials

Definition (Cyclotomic polynomial)

For any positive integer n, the nth *cyclotomic polynomial* is

$$\Phi_n(x) := (x - \omega_1)(x - \omega_2) \dots (x - \omega_n)$$

where $\omega_1, \dots, \omega_n$ are the primitive nth roots of unity, $\omega_k := e^{\frac{2\pi i}{n}k}$

Equivalently and less formally, the nth cyclotomic polynomial is the polynomial which:

- divides $x^n - 1$;
- does not divide $x^m - 1$ for any $m < n$;
- has integer coefficients;
- and is irreducible (cannot be factorised).

Fan & Vercauteren (2012) scheme : notation (cont'd)

- $\mathbb{Z}_q = \{n : n \in \mathbb{Z}, -q/2 < n \leq q/2\}$
- $[a]_q$ is unique integer in $\mathbb{Z}_q$ st $[a]_q = a \mod q$
- $\mathbb{Z}[x], \mathbb{Z}_q[x]$ denote polynomials with coefficients in $\mathbb{Z}$ and $\mathbb{Z}_q$ respectively
- $\Phi_n(x)$ is nth cyclotomic polynomial
- $\Phi_{2^d}(x) = x^{2^{d-1}} + 1$

Fan & Vercauteren (2012) scheme : notation (cont'd)

- $\mathbb{Z}_q = \{n : n \in \mathbb{Z}, -q/2 < n \leq q/2\}$
- $[a]_q$ is unique integer in $\mathbb{Z}_q$ st $[a]_q = a \mod q$
- $\mathbb{Z}[x], \mathbb{Z}_q[x]$ denote polynomials with coefficients in $\mathbb{Z}$ and $\mathbb{Z}_q$ respectively
- $\Phi_n(x)$ is nth cyclotomic polynomial
- $\Phi_{2^d}(x) = x^{2^{d-1}} + 1$
- Interest in elements of polynomial ring $R_q = \mathbb{Z}_q[x]/\Phi_{2^d}(x)$
- Polynomials written $\underline{a}$ or $a(x)$

Fan & Vercauteren (2012) scheme : notation (cont'd)

- $\mathbb{Z}_q = \{n : n \in \mathbb{Z}, -q/2 < n \leq q/2\}$
- $[a]_q$ is unique integer in $\mathbb{Z}_q$ st $[a]_q = a \mod q$
- $\mathbb{Z}[x], \mathbb{Z}_q[x]$ denote polynomials with coefficients in $\mathbb{Z}$ and $\mathbb{Z}_q$ respectively
- $\Phi_n(x)$ is nth cyclotomic polynomial
- $\Phi_{2^d}(x) = x^{2^{d-1}} + 1$
- Interest in elements of polynomial ring $R_q = \mathbb{Z}_q[x]/\Phi_{2^d}(x)$
- Polynomials written $\underline{a}$ or $a(x)$
- $\underline{a} \sim R_q \implies$ uniform random draw from R_q
- $\underline{a} \sim \chi \implies$ discrete multivariate Gaussian draw in R_q

Fan & Vercauteren (2012) scheme : notation (cont'd)

- $\mathbb{Z}_q = \{n : n \in \mathbb{Z}, -q/2 < n \leq q/2\}$
- $[a]_q$ is unique integer in $\mathbb{Z}_q$ st $[a]_q = a \mod q$
- $\mathbb{Z}[x], \mathbb{Z}_q[x]$ denote polynomials with coefficients in $\mathbb{Z}$ and $\mathbb{Z}_q$ respectively
- $\Phi_n(x)$ is nth cyclotomic polynomial
- $\Phi_{2^d}(x) = x^{2^{d-1}} + 1$
- Interest in elements of polynomial ring $R_q = \mathbb{Z}_q[x]/\Phi_{2^d}(x)$
- Polynomials written $\underline{a}$ or $a(x)$
- $\underline{a} \sim R_q \implies$ uniform random draw from R_q
- $\underline{a} \sim \chi \implies$ discrete multivariate Gaussian draw in R_q

Messages $m(x) \in M \triangleq R_t$

Cipher texts $c \in C \triangleq R_q \times R_q$

Fan & Vercauteren (2012) scheme : setup

- **Parameters**
  - d, degree of both the polynomial rings M and C
  - t and q, coefficient sets of polynomial rings M and C
  - σ, magnitude of the discrete Gaussian randomness for semantic security

Fan & Vercauteren (2012) scheme : setup

- **Parameters**
  - d, degree of both the polynomial rings M and C
  - t and q, coefficient sets of polynomial rings M and C
  - σ, magnitude of the discrete Gaussian randomness for semantic security
- **Key generation**
  - Secret key:

    $$\underline{k}_s \sim R_2$$

    (i.e. sample a 2^{d-1} binary vector for the polynomial coefficients).
  - Public key:

    $$k_p := ([-(\underline{a} \cdot \underline{k}_s + \underline{e})]_q, \underline{a})$$

    where $\underline{a} \sim R_q$ and $\underline{e} \sim \chi$.
    ($\underline{k}_s$ hard to extract due to ring LWE hardness, see Lyubashevsky et al. 2010)

Fan & Vercauteren (2012) : encryption/decryption

- **Encode**
  Need $m \in \mathbb{Z}$ expressed as polynomial ring element. Write in b-bit binary representation, $m = \sum_{n=0}^{b-1} a_n 2^n$, then construct $\mathring{m}(x) = \sum_{n=0}^{2^{d-1}-1} a_n x^n \in R_t$ where $a_n = 0 \, \forall \, n \geq b$.

Fan & Vercauteren (2012) : encryption/decryption

- **Encode**
  Need $m \in \mathbb{Z}$ expressed as polynomial ring element. Write in b-bit binary representation, $m = \sum_{n=0}^{b-1} a_n 2^n$, then construct $\mathring{m}(x) = \sum_{n=0}^{2^{d-1}-1} a_n x^n \in R_t$ where $a_n = 0 \; \forall \; n \geq b$.
- **Encryption** $\mathsf{Enc}(k_p, m)$
  First encode $m \in \mathbb{Z}$ as $\underline{\mathring{m}} \in R_t$

$$c := ([\underline{k}_{p1} \cdot \underline{u} + \underline{e}_1 + \Delta \cdot \underline{\mathring{m}}]_q, [\underline{k}_{p2} \cdot \underline{u} + \underline{e}_2]_q)$$

where $\underline{u}, \underline{e}_1, \underline{e}_2 \sim \chi$ and $\Delta = \lfloor \frac{q}{t} \rceil$.

Fan & Vercauteren (2012) : encryption/decryption

- **Encode**
  Need $m \in \mathbb{Z}$ expressed as polynomial ring element. Write in b-bit binary representation, $m = \sum_{n=0}^{b-1} a_n 2^n$, then construct $\mathring{m}(x) = \sum_{n=0}^{2^{d-1}-1} a_n x^n \in R_t$ where $a_n = 0 \; \forall \; n \geq b$.
- **Encryption** $\mathtt{Enc}(k_p, m)$
  First encode $m \in \mathbb{Z}$ as $\underline{\mathring{m}} \in R_t$

$$c := ([\underline{k}_{p1} \cdot \underline{u} + \underline{e}_1 + \Delta \cdot \underline{\mathring{m}}]_q, [\underline{k}_{p2} \cdot \underline{u} + \underline{e}_2]_q)$$

  where $\underline{u}, \underline{e}_1, \underline{e}_2 \sim \chi$ and $\Delta = \lfloor \frac{q}{t} \rceil$.
- **Decryption** $\mathtt{Dec}(k_s, c)$

$$\underline{\mathring{m}} = \left[\left\lfloor \frac{t[\underline{c}_1 + \underline{c}_2 \cdot \underline{k}_s]_q}{q} \right\rceil \right]_t$$

  so that $m = \mathring{m}(2)$ ... note, bootstrappable.

Fan & Vercauteren (2012) : understanding

$$\begin{aligned}&\mathtt{Dec}(k_s, c)\\ &= \left[\left\lfloor \frac{t[\underline{c}_1 + \underline{c}_2 \cdot \underline{k}_s]_q}{q} \right\rceil\right]_t\end{aligned}$$

Fan & Vercauteren (2012) : understanding

$$
\begin{aligned}
&\mathtt{Dec}(k_s, c) \\
&= \left[\left\lfloor \frac{t[\underline{c}_1 + \underline{c}_2 \cdot \underline{k}_s]_q}{q} \right\rceil\right]_t \\
&= \left[\left\lfloor \frac{t[\underline{k}_{p1} \cdot \underline{u} + \underline{e}_1 + \Delta \cdot \underline{\mathring{m}} + (\underline{k}_{p2} \cdot \underline{u} + \underline{e}_2) \cdot \underline{k}_s]_q}{q} \right\rceil\right]_t
\end{aligned}
$$

Fan & Vercauteren (2012) : understanding

$$
\begin{aligned}
&\mathtt{Dec}(k_s, c) \\
&= \left[\left\lfloor \frac{t[\underline{c}_1 + \underline{c}_2 \cdot \underline{k}_s]_q}{q} \right\rceil\right]_t \\
&= \left[\left\lfloor \frac{t[\underline{k}_{p1} \cdot \underline{u} + \underline{e}_1 + \Delta \cdot \underline{\mathring{m}} + (\underline{k}_{p2} \cdot \underline{u} + \underline{e}_2) \cdot \underline{k}_s]_q}{q} \right\rceil\right]_t \\
&= \left[\left\lfloor \frac{t[-(\underline{a} \cdot \underline{k}_s + \underline{e}) \cdot \underline{u} + \underline{e}_1 + \Delta \cdot \underline{\mathring{m}} + (\underline{a} \cdot \underline{u} + \underline{e}_2) \cdot \underline{k}_s]_q}{q} \right\rceil\right]_t
\end{aligned}
$$

Fan & Vercauteren (2012) : understanding

$$
\begin{aligned}
&\mathtt{Dec}(k_s, c) \\
&= \left[\left\lfloor \frac{t[\underline{c}_1 + \underline{c}_2 \cdot \underline{k}_s]_q}{q} \right\rceil\right]_t \\
&= \left[\left\lfloor \frac{t[\underline{k}_{p1} \cdot \underline{u} + \underline{e}_1 + \Delta \cdot \underline{\mathring{m}} + (\underline{k}_{p2} \cdot \underline{u} + \underline{e}_2) \cdot \underline{k}_s]_q}{q} \right\rceil\right]_t \\
&= \left[\left\lfloor \frac{t[-(\underline{a} \cdot \underline{k}_s + \underline{e}) \cdot \underline{u} + \underline{e}_1 + \Delta \cdot \underline{\mathring{m}} + (\underline{a} \cdot \underline{u} + \underline{e}_2) \cdot \underline{k}_s]_q}{q} \right\rceil\right]_t \\
&= \left[\left\lfloor \frac{t[-\underline{e} \cdot \underline{u} + \underline{e}_1 + \left\lfloor \frac{q}{t} \right\rceil \underline{\mathring{m}} + \underline{e}_2 \cdot \underline{k}_s]_q}{q} \right\rceil\right]_t
\end{aligned}
$$

Fan & Vercauteren (2012) : understanding

$$
\begin{aligned}
&\mathtt{Dec}(k_s, c) \\
&= \left[\left\lfloor \frac{t[\underline{c}_1 + \underline{c}_2 \cdot \underline{k}_s]_q}{q} \right\rceil\right]_t \\
&= \left[\left\lfloor \frac{t[\underline{k}_{p1} \cdot \underline{u} + \underline{e}_1 + \Delta \cdot \underline{\mathring{m}} + (\underline{k}_{p2} \cdot \underline{u} + \underline{e}_2) \cdot \underline{k}_s]_q}{q} \right\rceil\right]_t \\
&= \left[\left\lfloor \frac{t[-(\underline{a} \cdot \underline{k}_s + \underline{e}) \cdot \underline{u} + \underline{e}_1 + \Delta \cdot \underline{\mathring{m}} + (\underline{a} \cdot \underline{u} + \underline{e}_2) \cdot \underline{k}_s]_q}{q} \right\rceil\right]_t \\
&= \left[\left\lfloor \frac{t[-\underline{e} \cdot \underline{u} + \underline{e}_1 + \left\lfloor \frac{q}{t} \right\rceil \underline{\mathring{m}} + \underline{e}_2 \cdot \underline{k}_s]_q}{q} \right\rceil\right]_t
\end{aligned}
$$

Fan & Vercauteren (2012) : understanding

$$
\begin{aligned}
&\mathtt{Dec}(k_s, c) \\
&= \left[\left\lfloor \frac{t[\underline{c}_1 + \underline{c}_2 \cdot \underline{k}_s]_q}{q} \right\rceil\right]_t \\
&= \left[\left\lfloor \frac{t[\underline{k}_{p1} \cdot \underline{u} + \underline{e}_1 + \Delta \cdot \underline{\mathring{m}} + (\underline{k}_{p2} \cdot \underline{u} + \underline{e}_2) \cdot \underline{k}_s]_q}{q} \right\rceil\right]_t \\
&= \left[\left\lfloor \frac{t[-(\underline{a} \cdot \underline{k}_s + \underline{e}) \cdot \underline{u} + \underline{e}_1 + \Delta \cdot \underline{\mathring{m}} + (\underline{a} \cdot \underline{u} + \underline{e}_2) \cdot \underline{k}_s]_q}{q} \right\rceil\right]_t \\
&= \left[\left\lfloor \frac{t[-\underline{e} \cdot \underline{u} + \underline{e}_1 + \lfloor \frac{q}{t} \rceil \underline{\mathring{m}} + \underline{e}_2 \cdot \underline{k}_s]_q}{q} \right\rceil\right]_t
\end{aligned}
$$

But, note that $\|-\underline{e} \cdot \underline{u} + \underline{e}_1 + \underline{e}_2 \cdot \underline{k}_s\|_\infty \ll \frac{q}{t}$ by construction, so that after multiplication by $\frac{t}{q}$ the only term surviving rounding is $\underline{\mathring{m}}$.

Fan & Vercauteren (2012) : addition/multiplication

- **Addition**, $+$ Standard vector and polynomial addition with modulo reduction:

$$c_1 + c_2 = ([\underline{c}_{11} + \underline{c}_{21}]_q, [\underline{c}_{12} + \underline{c}_{22}]_q)$$

- **Multiplication** $\times$ Multiplication increases length of the cipher text vector:

$$c_1 \times c_2 = \left(\left[\left\lfloor \frac{t(\underline{c}_{11} \cdot \underline{c}_{21})}{q} \right\rceil \right]_q, \left[\left\lfloor \frac{t(\underline{c}_{11} \cdot \underline{c}_{22} + \underline{c}_{12} \cdot \underline{c}_{21})}{q} \right\rceil \right]_q, \left[\left\lfloor \frac{t(\underline{c}_{12} \cdot \underline{c}_{22})}{q} \right\rceil \right]_q \right)$$

Still possible to recover $\underline{\mathring{m}}$ by modifying decryption to be $\left[\left\lfloor \frac{t}{q} [\underline{c}_1 + \underline{c}_2 \cdot \underline{k}_s + \underline{c}_3 \cdot \underline{k}_s \cdot \underline{k}_s]_q \right\rceil \right]_t$, it is preferable to perform a 'relinearisation' procedure which compacts the cipher text to a vector of two polynomials again.

Some important points to note

Operations on messages, m, are really operating on the encoding, $\mathring{m}(x) = \sum_{n=0}^{2^{d-1}-1} a_n x^n \in R_t$. So, encoding is in R_t

Some important points to note

Operations on messages, m, are really operating on the encoding, $\mathring{m}(x) = \sum_{n=0}^{2^{d-1}-1} a_n x^n \in R_t$. So, encoding is in R_t $\Longrightarrow$

- the coefficients of the underlying $\mathring{m}(x)$ must lie within $\left(\frac{t}{2}, \frac{t}{2}\right]$;
- the order of $\mathring{m}(x)$ must not exceed $2^{d-1} - 1$.

Since we always start from a binary encoding, we can often prove theoretical bounds for a given algorithm executing correctly.

Some important points to note

Operations on messages, m, are really operating on the encoding, $\mathring{m}(x) = \sum_{n=0}^{2^{d-1}-1} a_n x^n \in R_t$. So, encoding is in R_t $\Longrightarrow$

- the coefficients of the underlying $\mathring{m}(x)$ must lie within $(\frac{t}{2}, \frac{t}{2}]$;
- the order of $\mathring{m}(x)$ must not exceed $2^{d-1} - 1$.

Since we always start from a binary encoding, we can often prove theoretical bounds for a given algorithm executing correctly.

Toy example: $d = 32, q = 2^{24} = 16777216, t = 32, \sigma = 3$.

Demo: 003_HEdeg.R

Parameter choice

A reasonable default of:

$d = 4096$

$q = 2^{128} = 340282366920938463463374607431768211456$

$t = 32768$

$\sigma = 16$

gives approximately 128-bit security level and about 4 multiplications deep.

Parameter choice

There are theoretical bounds on both multiplicative depth and security level in the literature (Lindner & Peikert (2011), Fan & Vercauteren (2012), Lepoint & Naehrig (2014))

Theorem (Fan & Vercauteren 2012)

Given the leveled homomorphic encryption scheme FV with parameters d, q, t and σ, the maximum multiplicative depth $L_{\max}$ supported satisfies:

$$4\,\beta(\varepsilon)\,\delta_R^{L_{\max}}\,(\delta_R + 1.25)^{L_{\max}+1}\,t^{L_{\max}-1} < \frac{q}{\sigma}$$

Limitations overview

1. Message space
   - R_t, so must encode single datum as polynomials
2. Cipher text size
   - Single 4/8-byte value $\in \mathbb{Z}$ transformed to $R_q \times R_q \implies$ 128KB for parameters on previous slide
3. Computational cost
   - 1 message $+ \implies$ 8192 lots of 128-bit modular addition
   - 1 message $\times \implies$ 4 lots of 4096 degree polynomial multiplcations involving 128-bit values, plus 8192 lots of 128-bit addition, plus integer addition and multiplication followed by polynomial modular reduction.
4. Division and comparison operations
   - Impossible!
5. Depth of operations
   - multiplications limited because end up with products of $-\underline{e} \cdot \underline{u}, \underline{e}_1$ and $\underline{e}_2$ terms so that ultimately noise exceeds $\frac{q}{t}$

Ameliorating computational burden

Theorem (Chinese Remainder Theorem)

Let $m_1, \ldots, m_k \in \mathbb{Z}^+$ be pairwise coprime positive integers. Let $M = \prod_{i=1}^{k} m_i$ and let $x_1, \ldots, x_k \in \mathbb{Z}$. Then there is exactly one integer x that satisfies the conditions:

$$0 \leq x < M \qquad \textit{and} \qquad x \equiv x_i \mod m_i \ \forall\, 1 \leq i \leq k$$

Thus, an integer message $x \in [0, M)$ can be uniquely represented by the collection of smaller integers $\{x_i\}_{i=1}^{k}$ ... this is a Residue Number System. Conversely, can also think of $\{x_i\}_{i=1}^{k}$ being represented by x.

Going $x \to \{x_i\}_{i=1}^{k}$ is simply taking modulo each m_i.

Going $\{x_i\}_{i=1}^{k} \to x$ can be constructed via the extended Euclidean algorithm.

Arithmetic with CRT

In particular, note that a Chinese Remainder Theorem representation preserves modular arithmetic.

Let $\{x_i\}_{i=1}^k, \{y_i\}_{i=1}^k$ be two collections of residue numbers, modulo $\{m_i\}_{i=1}^k$. Let x and y be the corresponding integers satisfying the Chinese Remainder Theorem. Then,

$$z = x + y \iff z \mod m_i = z_i = (x_i + y_i) \mod m_i$$

In other words, doing one addition $(x + y)$ actually gives k additions by looking at the single result modulo each m_i.

Arithmetic with CRT

In particular, note that a Chinese Remainder Theorem representation preserves modular arithmetic.

Let $\{x_i\}_{i=1}^k, \{y_i\}_{i=1}^k$ be two collections of residue numbers, modulo $\{m_i\}_{i=1}^k$. Let x and y be the corresponding integers satisfying the Chinese Remainder Theorem. Then,

$$z = x + y \iff z \mod m_i = z_i = (x_i + y_i) \mod m_i$$

In other words, doing one addition $(x + y)$ actually gives k additions by looking at the single result modulo each m_i.

Demo: 001_CRT.R

Polynomial Chinese Remainder Theorem (I)

There is a corresponding CRT for polynomials!

Although $\Phi_n(x)$ is irreducible over $\mathbb{Q}[x]$, it is not necessarily irreducible over $\mathbb{Z}_t[x]$. Suppose it has r factors:

$$\Phi_n(x) = \prod_{j=1}^{r} f_j(x)$$

Then, we can encode a vector of polynomial messages $(\mathring{\underline{m}}_1, \ldots, \mathring{\underline{m}}_r)$ since by the Polynomial Chinese Remainder Theorem $\exists\, \underline{m} \in \mathbb{Z}_t[x]/\Phi_n(x)$ such that $\underline{m} \mod f_i(x) = \mathring{\underline{m}}_i$.

Upshot: if we now encrypt $\underline{m}$, then we have encrypted a CRT representation of r messages in just one cipher text.

Polynomial Chinese Remainder Theorem (II)

So, consider a collection of vectors of polynomials encoded in this way

$$\mathbb{Z}_t[x]/f_1(x) \times \cdots \times \mathbb{Z}_t[x]/f_r(x) \ni (\underline{\mathring{m}}_{i1}, \ldots, \underline{\mathring{m}}_{ir}) \longrightarrow \underline{\mathring{m}}_i \in R_t$$

Then,

$$\left(\sum_i \underline{\mathring{m}}_i\right) \mod f_j(x) = \left(\sum_i \underline{\mathring{m}}_{ij}\right) \mod f_j(x) \quad \forall j = 1, \ldots, r$$

$$\left(\prod_i \underline{\mathring{m}}_i\right) \mod f_j(x) = \left(\prod_i \underline{\mathring{m}}_{ij}\right) \mod f_j(x) \quad \forall j = 1, \ldots, r$$

In other words, we can do SIMD on cipher texts. There also exist automorphism mappings which will allow slots to be exchanged and interacted. (Smart & Vercauteren 2014)

Software

Existing implementations

- `libfhe` (Minar 2010) compact single C file library implementing Gentry (2010)
- 'Scarab' (Perl et al. 2011) low level C library implementing Smart & Vercauteren (2010)
- 'HELib' (Halevi & Shoup 2014) most impressive library, in C++ implementing Brakerski et al. (2012) and lots beyond the bare bones cryptography (i.e. Polynomial Chinese Remainder Theorem + automorphisms)
- more besides ...

However, these all tend to be very low-level libraries.

HomomorphicEncryption R package (Aslett 2014)

All core code in high-performance multi-threaded C++, but accessible via simple R functions and overloaded operators:

```
library("HomomorphicEncryption")

p <- pars("FandV")
k <- keygen(p)
c1 <- enc(k$pk, c(42,34))
c2 <- enc(k$pk, c(7,5))
cres1 <- c1 + c2
cres2 <- c1 * c2
cres3 <- c1 %*% c2
dec(k$sk, cres1)
dec(k$sk, cres2)
dec(k$sk, cres3)
```

Demo: 002_FHE.R

Multiparty Computing

Multiparty Computing

Homomorphic encryption allows private computation with security guaranteed to one individual.

What if multiple parties want to cooperate, pool their data for computation while retaining security guarantees on the original data?

- Can this be solved?
- How strong are the security guarantees?
- Can we prevent cheating?

Multiparty Computing

Homomorphic encryption allows private computation with security guaranteed to one individual.

What if multiple parties want to cooperate, pool their data for computation while retaining security guarantees on the original data?

- Can this be solved?
- How strong are the security guarantees?
- Can we prevent cheating?

NB notation change: $\mathbb{Z}_p$ denotes the standard ring of integers modulo p.

Simple Homomorphic Secret Sharing

Party $i \in \{1, 2, 3\}$ wants to share secret $s_i \in \mathbb{Z}_p$.

Each party will:

1. Choose $r_{i1}, r_{i2} \in \mathbb{Z}_p$ uniformly at random.
2. Compute $r_{i3} = (s_i - r_{i1} - r_{i2}) \mod p$
3. Send $r_{i,-j}$ to party j
   - party 1 has $r_{i2}, r_{i3} \; \forall \; i$
   - party 2 has $r_{i1}, r_{i3} \; \forall \; i$
   - party 3 has $r_{i1}, r_{i2} \; \forall \; i$

Note that $s_i = (r_{i1} + r_{i2} + r_{i3}) \mod p$, but no party (except i) has more than 2 of these values. They are unformly random and hence uninformative about s_i.

Simple Homomorphic Secret Sharing : operations

- **Addition**, $+$
  Each party i:
  - Perform steps on previous slide to distribute shares
  - Computes $v_l = r_{1l} + r_{2l} + r_{3l}$ for $l \neq i$
  - Sends v_l to other parties
  - All compute $(v_1 + v_2 + v_3) \mod p \equiv (s_1 + s_2 + s_3) \mod p$

Simple Homomorphic Secret Sharing : operations

- **Addition**, $+$
  Each party i:
  - Perform steps on previous slide to distribute shares
  - Computes $v_l = r_{1l} + r_{2l} + r_{3l}$ for $l \neq i$
  - Sends v_l to other parties
  - All compute $(v_1 + v_2 + v_3) \mod p \equiv (s_1 + s_2 + s_3) \mod p$
- **Multiplication** $\times$
  Suppose we want $s_1 \times s_2$ (product of only party 1 and 2).
  Party 3 is a semi-trusted third party.
  - Party 1 and 2 perform steps on previous slide to distribute shares
  - Each party computes:
    - Party 1: $v_1 = r_{12}r_{22} + r_{12}r_{23} + r_{13}r_{22} \mod p$
    - Party 2: $v_2 = r_{13}r_{23} + r_{11}r_{23} + r_{13}r_{21} \mod p$
    - Party 3: $v_3 = r_{11}r_{21} + r_{11}r_{22} + r_{12}r_{21} \mod p$
  - All parties create secret shares of v_i and cooperatively compute $v_1 + v_2 + v_3 \mod p \equiv s_1 s_2 \mod p$

Simple Homomorphic Secret Sharing : security

How secure is this secret sharing compared to homomorphic encryption?

Assuming parties do not collude:

Information Theoretically Secure

This means that an adversary with *unbounded* compute power cannot determine your secret data.

Simple Homomorphic Secret Sharing : comments

- Can we detect deviation from the agreed computation?
  - Addition: yes, all parties compute two values ($v_j, j \neq i$) so disagreement reveals dishonesty when they are exchanged.
  - Multiplication: not in the first step.

Simple Homomorphic Secret Sharing : comments

- Can we detect deviation from the agreed computation?
  - Addition: yes, all parties compute two values ($v_j, j \neq i$) so disagreement reveals dishonesty when they are exchanged.
  - Multiplication: not in the first step.
- Multiplication involves rounds of communication: this can be painful (moreso that the cost of homomorphic encryption in some settings)
  - Note the secure addition is neccessary to avoid a low probability event that would reveal information.

Simple Homomorphic Secret Sharing : comments

- Can we detect deviation from the agreed computation?
  - Addition: yes, all parties compute two values ($v_j, j \neq i$) so disagreement reveals dishonesty when they are exchanged.
  - Multiplication: not in the first step.
- Multiplication involves rounds of communication: this can be painful (moreso that the cost of homomorphic encryption in some settings)
  - Note the secure addition is neccessary to avoid a low probability event that would reveal information.
- Multiplication involves a third party who has nothing at stake.

Simple Homomorphic Secret Sharing : comments

- Can we detect deviation from the agreed computation?
  - Addition: yes, all parties compute two values ($v_j, j \neq i$) so disagreement reveals dishonesty when they are exchanged.
  - Multiplication: not in the first step.
- Multiplication involves rounds of communication: this can be painful (moreso that the cost of homomorphic encryption in some settings)
  - Note the secure addition is neccessary to avoid a low probability event that would reveal information.
- Multiplication involves a third party who has nothing at stake.
- These can be trivially extended beyond 3 parties.

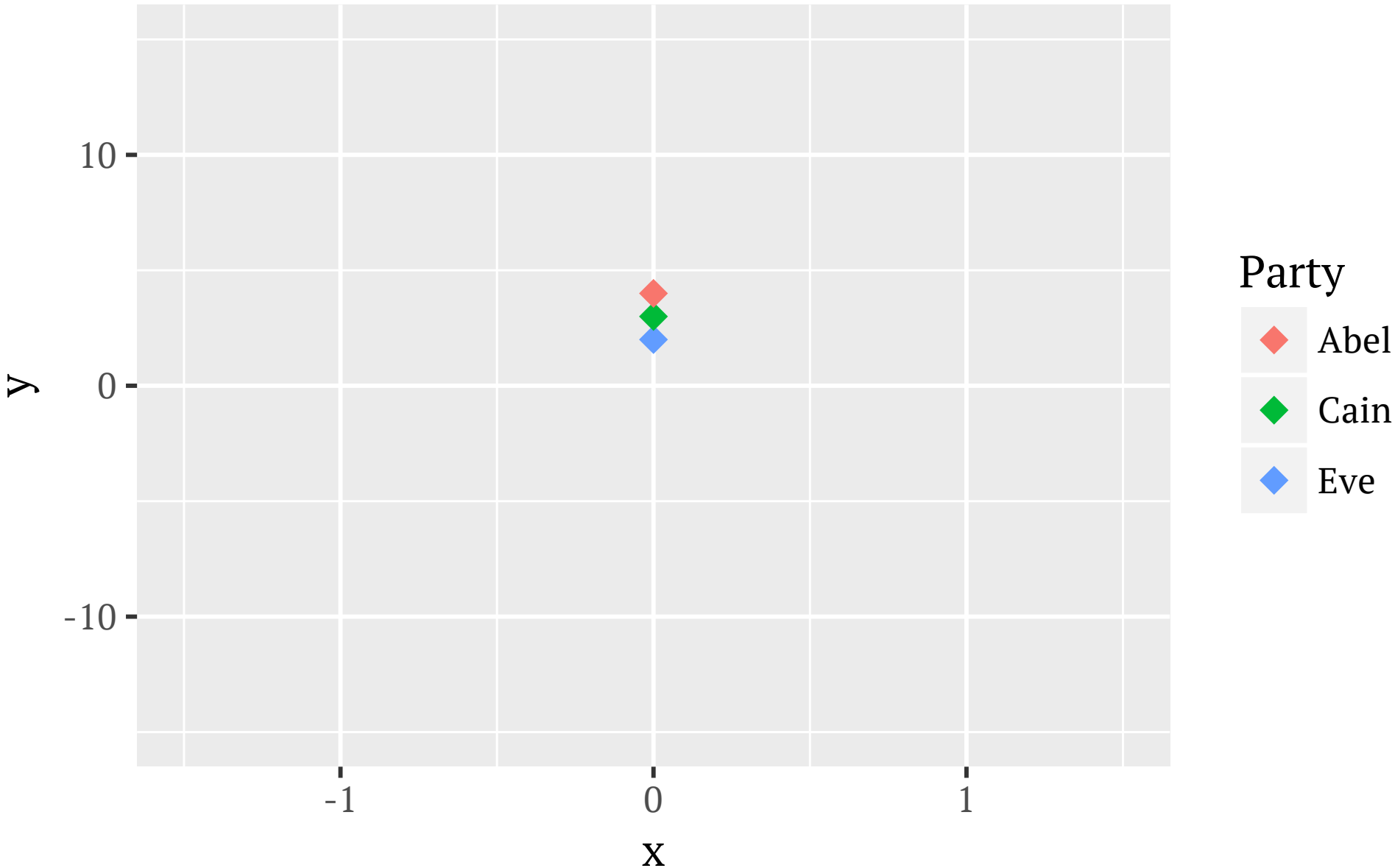
Advanced Homomorphic Secret Sharing

There are more advanced methods, based on polynomials over a finite field, where:

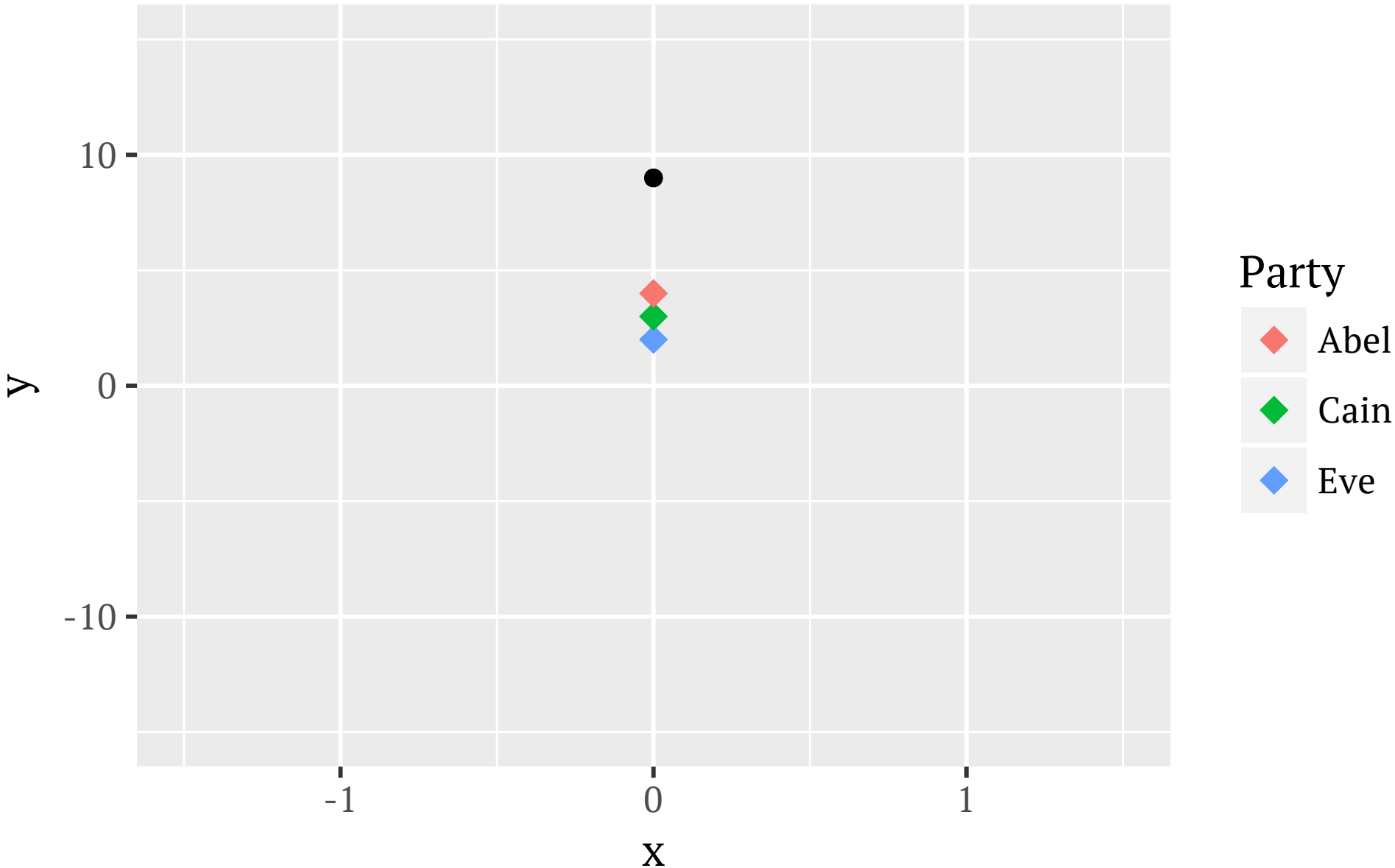
- Upto $\frac{1}{3}$ of parties may be corrupted.
- Combining cryptographic and secret sharing to manage dishonest majority scenarios (but losing information theoretic security).
- Security against active attackers.
- Both perfectly and imperfectly secure communication channels.
- ...

Could devote a whole 3 hour course to just a subset of these! We simply sketch an intuiton for the homomorphic properties of polynomial shares ...

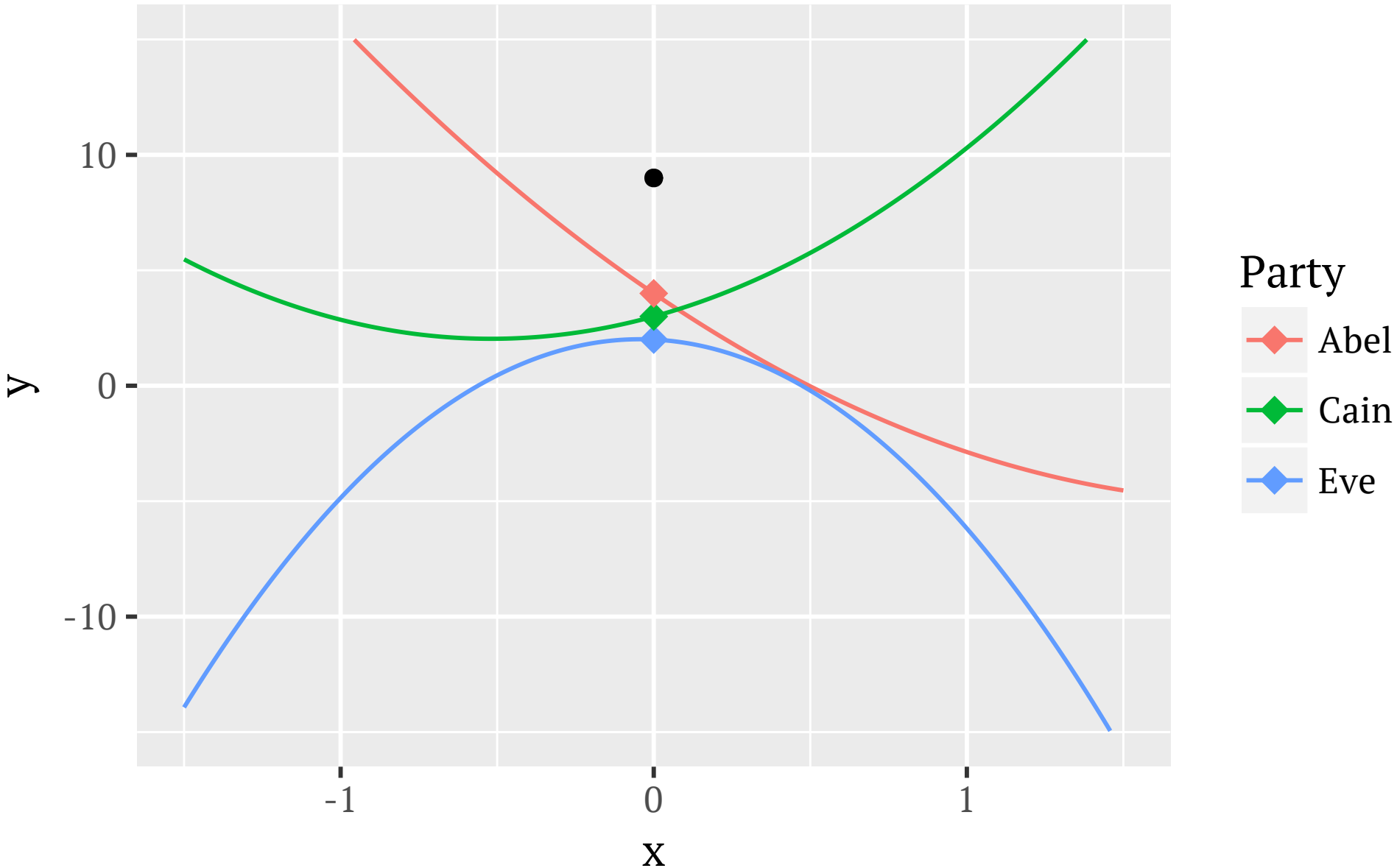
(Simplified) HSS /w Polynomials



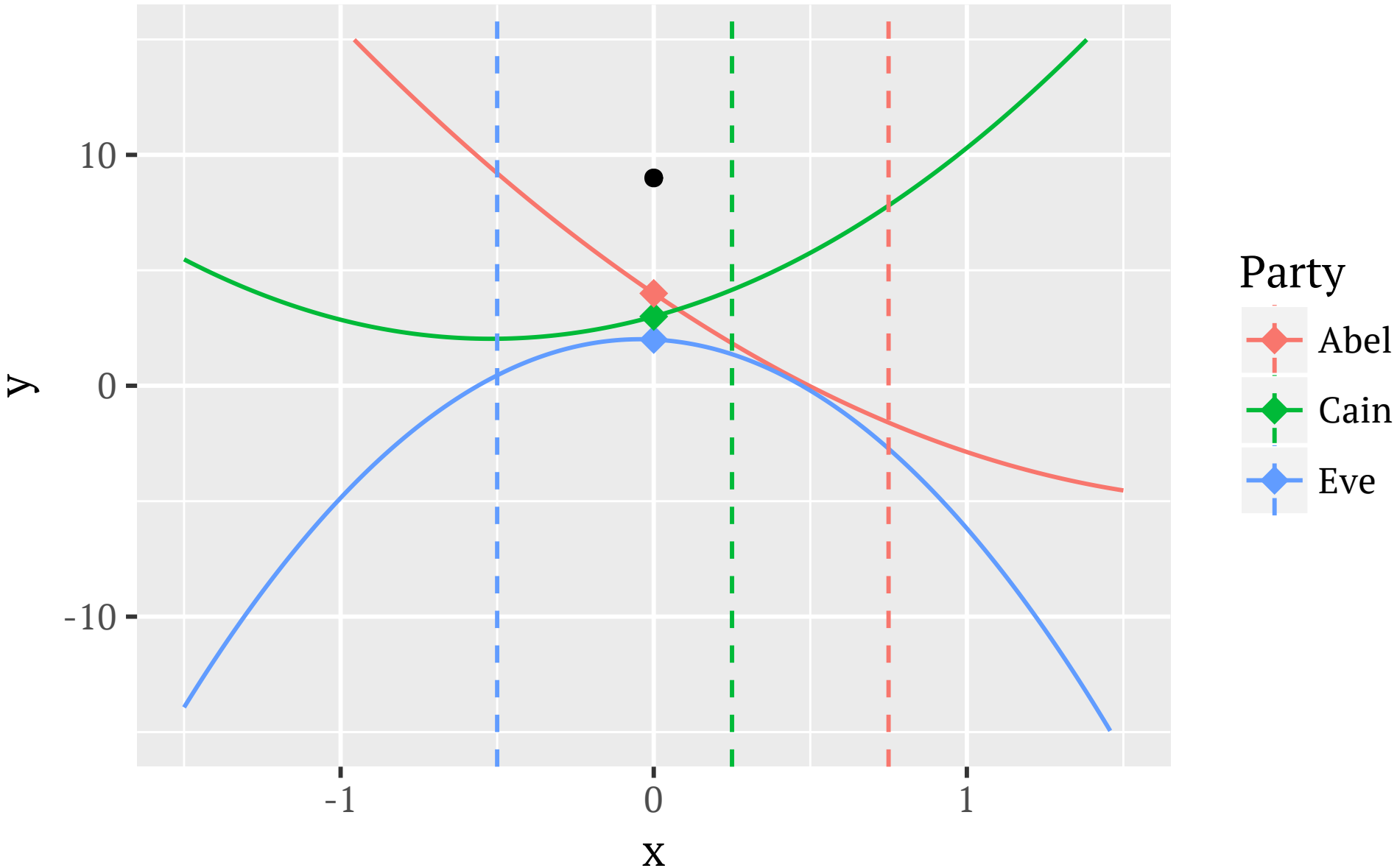
(Simplified) HSS /w Polynomials



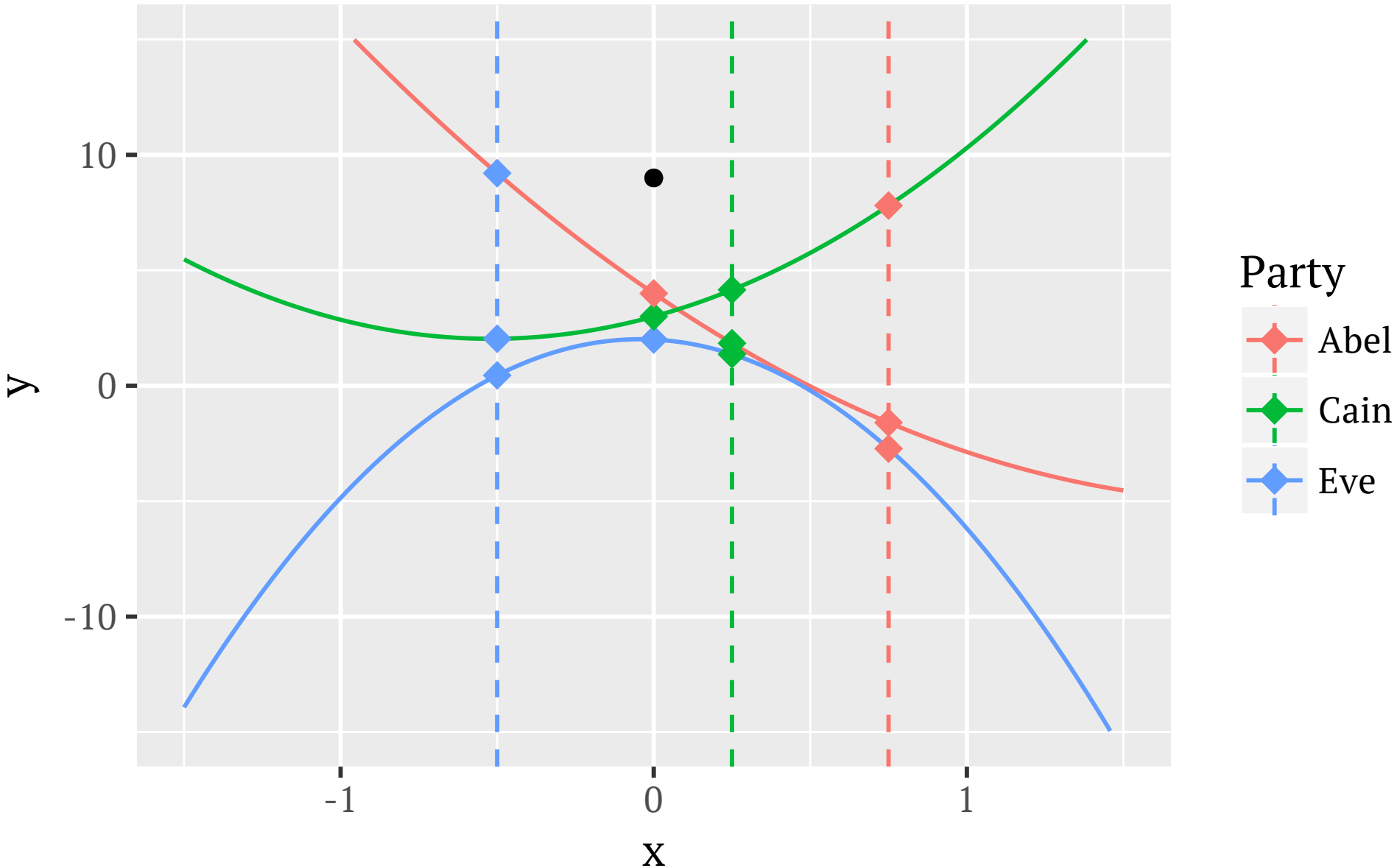
(Simplified) HSS /w Polynomials



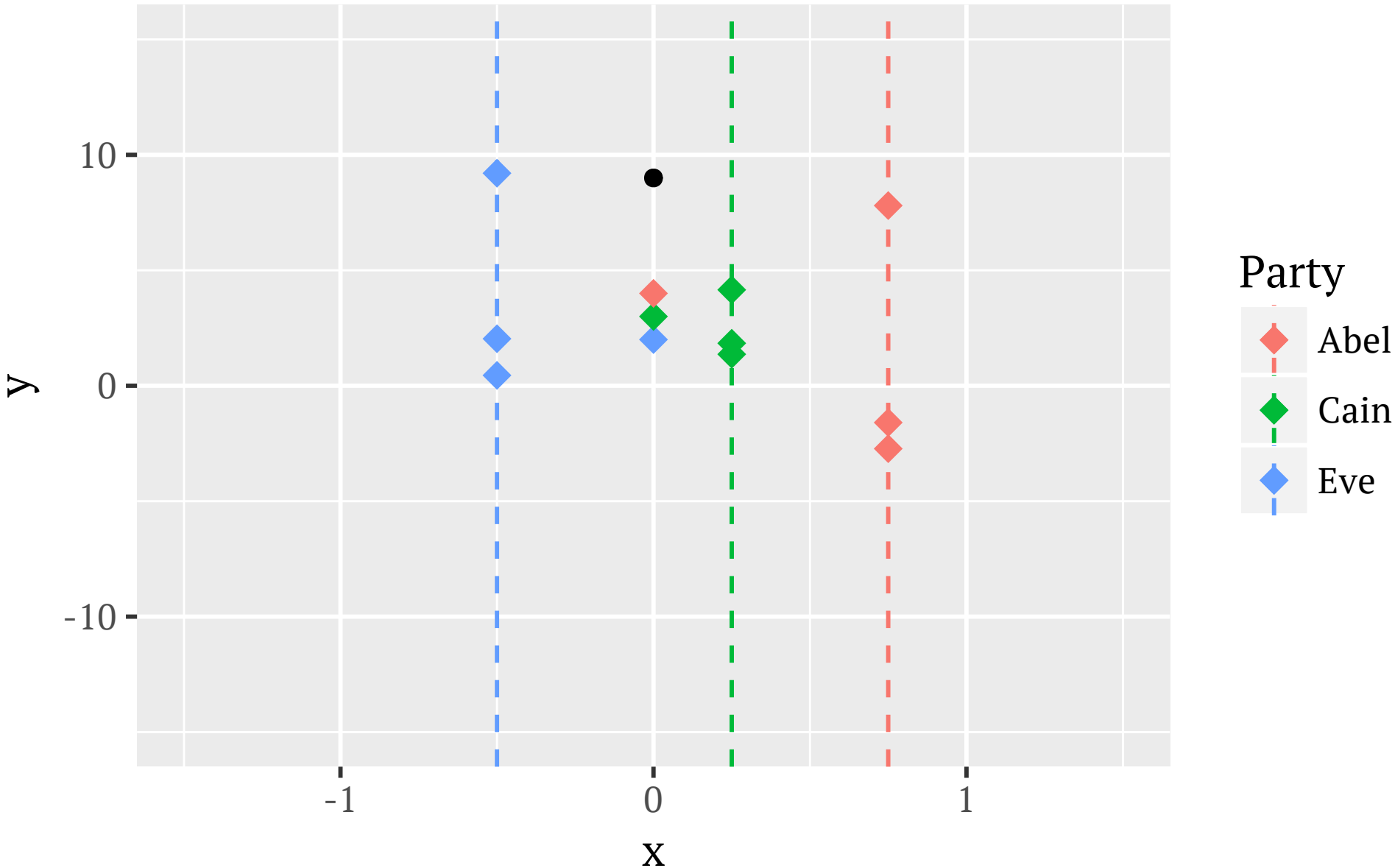
(Simplified) HSS /w Polynomials



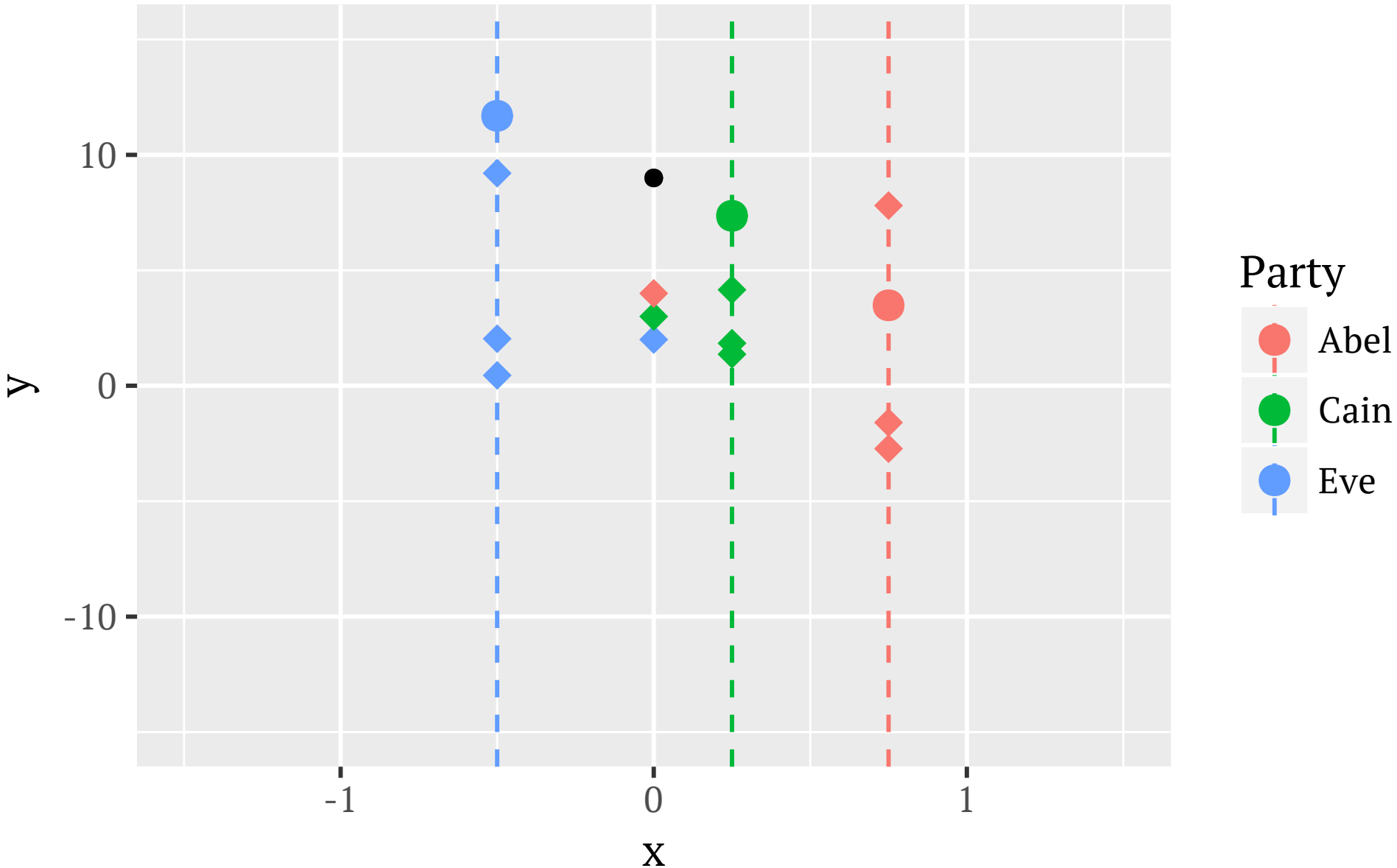
(Simplified) HSS /w Polynomials



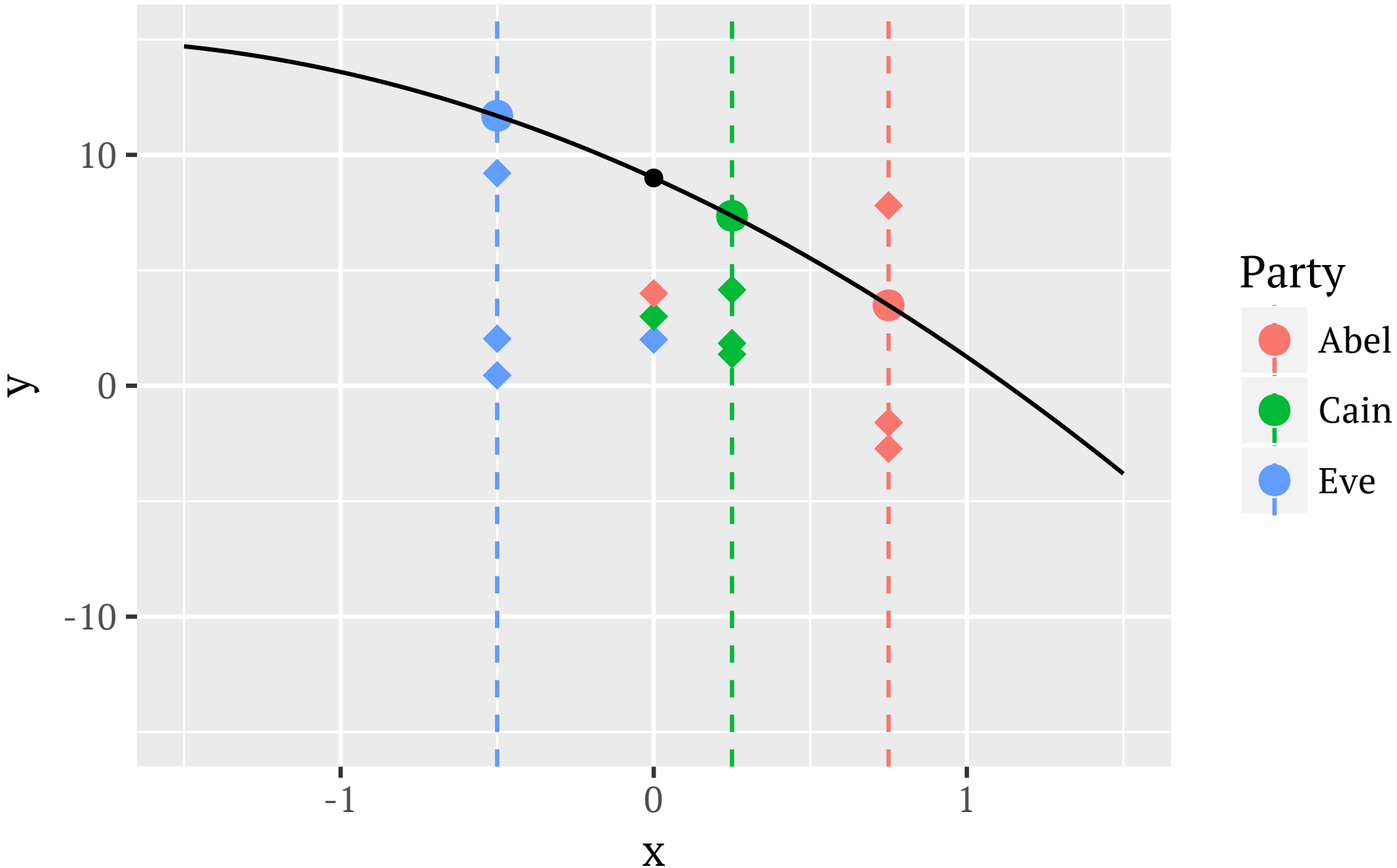
(Simplified) HSS /w Polynomials



(Simplified) HSS /w Polynomials



(Simplified) HSS /w Polynomials



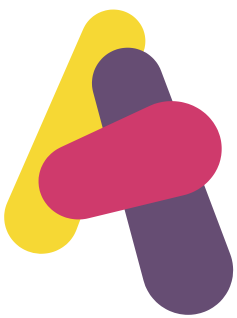
Shameless plug! Knowledge Transfer Partnership

Forthcoming KTP associate job, based at Atom Bank working with me and Camila Caiado at Durham University.

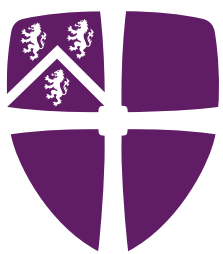
Jointly working with Computer Science KTP associate based at Atom and working with Newcastle University.

Statistical modelling and encrypted statistics for mortgage books.

Expected to advertise for an August – October 2018 start.



Atom bank



Durham University

References I

Aslett, L. J. M. (2014). HomomorphicEncryption: Fully homomorphic encryption. http://www.louisaslett.com/HomomorphicEncryption/.

Brakerski, Z., Gentry, C., & Vaikuntanathan, V. (2012). (Leveled) fully homomorphic encryption without bootstrapping. *Proceedings of the 3rd innovations in theoretical computer science conference*, pp. 309–25. ACM.

Fan, J., & Vercauteren, F. (2012). Somewhat practical fully homomorphic encryption. *IACR Cryptology ePrint Archive*.

Gentry, C. (2009). *A fully homomorphic encryption scheme* (PhD thesis). Stanford University. Retrieved from <crypto.stanford.edu/craig>

Gentry, C. (2010). Computing arbitrary functions of encrypted data. *Communications of the ACM*, 53/3: 97–105. ACM.

Halevi, S., & Shoup, V. (2014). HElib. https://github.com/shaih/HElib.

Lepoint, T., & Naehrig, M. (2014). A comparison of the homomorphic encryption schemes FV and YASHE. *Progress in cryptology–AFRICACRYPT*

References II

2014, pp. 318–35. Springer.

Lindner, R., & Peikert, C. (2011). Better key sizes (and attacks) for LWE-based encryption. *Topics in cryptology–CT-rsa 2011*, pp. 319–39. Springer.

Lyubashevsky, V., Peikert, C., & Regev, O. (2010). On ideal lattices and learning with errors over rings. *Proceedings of the 29th annual international conference on theory and applications of cryptographic techniques*. Springer-Verlag.

Minar, J. (2010). Libfhe. https://github.com/rdancer/fhe/tree/master/libfhe.

Perl, H., Brenner, M., & Smith, M. (2011). Scarab library. https://hcrypt.com/scarab-library/.

Rivest, R. L., Adleman, L., & Dertouzos, M. L. (1978). On data banks and privacy homomorphisms. *Foundations of Secure Computation*, 4/11: 169–80.

Smart, N. P., & Vercauteren, F. (2010). Fully homomorphic encryption with relatively small key and ciphertext sizes. *Public key cryptography–PKC 2010*,

References III

pp. 420–43. Springer.

Smart, N. P., & Vercauteren, F. (2014). Fully homomorphic SIMD operations. *Designs, codes and cryptography*, 71/1: 57–81.