Functional encryption and multivariate cryptography

Roberto La Scala

Università degli Studi di Bari “Aldo Moro”

CIFRIS24, Cloud, 27-09-2024

Motivation

- The compliance of Cloud Storage and Cloud Computing with recent privacy regulations (GDPR - Europe, HIPAA - USA, PIPEDA - Canada, LGPD - Brasil, etc.) requires that all computations are performed while preserving the confidentiality of user data. Here are a couple of case studies.

- Consider a bank that wants to enable a credit verification agency to search encrypted financial data of its customers without revealing the entire database. To achieve this, the bank aims to implement suitable cryptographic tools that permit only the computation of specific credit evaluation functions, returning a result (e.g., whether a customer meets a certain credit threshold) without exposing any other sensitive information.

Motivation

- The compliance of Cloud Storage and Cloud Computing with recent privacy regulations (GDPR - Europe, HIPAA - USA, PIPEDA - Canada, LGPD - Brasil, etc.) requires that all computations are performed while preserving the confidentiality of user data. Here are a couple of case studies.

- Consider a bank that wants to enable a credit verification agency to search encrypted financial data of its customers without revealing the entire database. To achieve this, the bank aims to implement suitable cryptographic tools that permit only the computation of specific credit evaluation functions, returning a result (e.g., whether a customer meets a certain credit threshold) without exposing any other sensitive information.

Motivation

- The compliance of Cloud Storage and Cloud Computing with recent privacy regulations (GDPR - Europe, HIPAA - USA, PIPEDA - Canada, LGPD - Brasil, etc.) requires that all computations are performed while preserving the confidentiality of user data. Here are a couple of case studies.
- Consider a bank that wants to enable a credit verification agency to search encrypted financial data of its customers without revealing the entire database. To achieve this, the bank aims to implement suitable cryptographic tools that permit only the computation of specific credit evaluation functions, returning a result (e.g., whether a customer meets a certain credit threshold) without exposing any other sensitive information.

- In a medical research context, a hospital might want to allow researchers to perform statistical analyses on encrypted health data while preserving patient privacy. By suitable encryption-decryption algorithms, the hospital can provide special keys that allow the calculation of specific aggregate statistics on the encrypted data without ever revealing individual medical records.
- The paradigm of *Functional Encryption* offers a robust solution to these and numerous other cryptographic challenges within the context of Cloud Computing.

- In a medical research context, a hospital might want to allow researchers to perform statistical analyses on encrypted health data while preserving patient privacy. By suitable encryption-decryption algorithms, the hospital can provide special keys that allow the calculation of specific aggregate statistics on the encrypted data without ever revealing individual medical records.
- The paradigm of *Functional Encryption* offers a robust solution to these and numerous other cryptographic challenges within the context of Cloud Computing.

Formal Definition

- Let X denote a set of *plaintexts* and Y a set of *ciphertexts*.
- Let K be a set of *functional keys*. We generally assume that K contains a special element denoted by λ. We call this element the *empty (functional) key*.
- A function $F : K \times X \to V$ is given where V denotes a set of *functional values*.
- Let K_s be a set of *secret keys*, let K_p be a set of *public keys* and let K_u be a set of *user secret keys*.
- A function $\mathrm{keygen} : K \times K_s \to K_u$ is given, that is, for any functional key $k \in K$ and for any secret key $sk \in K_s$, one has a user secret key $usk_k = \mathrm{keygen}(k, sk) \in K_u$.

Formal Definition

- Let X denote a set of *plaintexts* and Y a set of *ciphertexts*.
- Let K be a set of *functional keys*. We generally assume that K contains a special element denoted by λ. We call this element the *empty (functional) key*.
- A function $F : K \times X \to V$ is given where V denotes a set of *functional values*.
- Let K_s be a set of *secret keys*, let K_p be a set of *public keys* and let K_u be a set of *user secret keys*.
- A function $\mathrm{keygen} : K \times K_s \to K_u$ is given, that is, for any functional key $k \in K$ and for any secret key $sk \in K_s$, one has a user secret key $usk_k = \mathrm{keygen}(k, sk) \in K_u$.

Formal Definition

- Let X denote a set of *plaintexts* and Y a set of *ciphertexts*.
- Let K be a set of *functional keys*. We generally assume that K contains a special element denoted by λ. We call this element the *empty (functional) key*.
- A function $F : K \times X \to V$ is given where V denotes a set of *functional values*.
- Let K_s be a set of *secret keys*, let K_p be a set of *public keys* and let K_u be a set of *user secret keys*.
- A function $\mathrm{keygen} : K \times K_s \to K_u$ is given, that is, for any functional key $k \in K$ and for any secret key $sk \in K_s$, one has a user secret key $usk_k = \mathrm{keygen}(k, sk) \in K_u$.

Formal Definition

- Let X denote a set of *plaintexts* and Y a set of *ciphertexts*.
- Let K be a set of *functional keys*. We generally assume that K contains a special element denoted by λ. We call this element the *empty (functional) key*.
- A function $F : K \times X \to V$ is given where V denotes a set of *functional values*.
- Let K_s be a set of *secret keys*, let K_p be a set of *public keys* and let K_u be a set of *user secret keys*.
- A function $\mathrm{keygen} : K \times K_s \to K_u$ is given, that is, for any functional key $k \in K$ and for any secret key $sk \in K_s$, one has a user secret key $usk_k = \mathrm{keygen}(k, sk) \in K_u$.

Formal Definition

- Let X denote a set of *plaintexts* and Y a set of *ciphertexts*.
- Let K be a set of *functional keys*. We generally assume that K contains a special element denoted by λ. We call this element the *empty (functional) key*.
- A function $F : K \times X \to V$ is given where V denotes a set of *functional values*.
- Let K_s be a set of *secret keys*, let K_p be a set of *public keys* and let K_u be a set of *user secret keys*.
- A function $\mathrm{keygen} : K \times K_s \to K_u$ is given, that is, for any functional key $k \in K$ and for any secret key $sk \in K_s$, one has a user secret key $usk_k = \mathrm{keygen}(k, sk) \in K_u$.

Formal Definition

- Let X denote a set of *plaintexts* and Y a set of *ciphertexts*.
- Let K be a set of *functional keys*. We generally assume that K contains a special element denoted by λ. We call this element the *empty (functional) key*.
- A function $F : K \times X \to V$ is given where V denotes a set of *functional values*.
- Let K_s be a set of *secret keys*, let K_p be a set of *public keys* and let K_u be a set of *user secret keys*.
- A function $\mathrm{keygen} : K \times K_s \to K_u$ is given, that is, for any functional key $k \in K$ and for any secret key $sk \in K_s$, one has a user secret key $usk_k = \mathrm{keygen}(k, sk) \in K_u$.

- For any public key $pk \in K_p$, we have the *functional encryption algorithm* $\mathrm{enc}_{pk} : X \to Y$.
- For any user secret key usk_k, we have the *functional decryption algorithm* $\mathrm{dec}_{usk_k} : Y \to V$ such that if $y = \mathrm{enc}_{pk}(x)$ then $\mathrm{dec}_{usk_k}(y) = F(k, x)$.
- Finally, we have a *setup function* that, given a suitable choice of parameters, returns a random pair consisting of a secret key and a public key, generated in accordance with these parameters.
- The set of all these functions defines a protocol of *Functional Encryption*. We assume that they can be computed efficiently.

- For any public key $pk \in K_p$, we have the *functional encryption algorithm* $\mathrm{enc}_{pk} : X \to Y$.
- For any user secret key usk_k, we have the *functional decryption algorithm* $\mathrm{dec}_{usk_k} : Y \to V$ such that if $y = \mathrm{enc}_{pk}(x)$ then $\mathrm{dec}_{usk_k}(y) = F(k, x)$.
- Finally, we have a *setup function* that, given a suitable choice of parameters, returns a random pair consisting of a secret key and a public key, generated in accordance with these parameters.
- The set of all these functions defines a protocol of *Functional Encryption*. We assume that they can be computed efficiently.

- For any public key $pk \in K_p$, we have the *functional encryption algorithm* $\mathrm{enc}_{pk} : X \to Y$.
- For any user secret key usk_k, we have the *functional decryption algorithm* $\mathrm{dec}_{usk_k} : Y \to V$ such that if $y = \mathrm{enc}_{pk}(x)$ then $\mathrm{dec}_{usk_k}(y) = F(k, x)$.
- Finally, we have a *setup function* that, given a suitable choice of parameters, returns a random pair consisting of a secret key and a public key, generated in accordance with these parameters.
- The set of all these functions defines a protocol of *Functional Encryption*. We assume that they can be computed efficiently.

- For any public key $pk \in K_p$, we have the *functional encryption algorithm* $\mathrm{enc}_{pk} : X \to Y$.
- For any user secret key usk_k, we have the *functional decryption algorithm* $\mathrm{dec}_{usk_k} : Y \to V$ such that if $y = \mathrm{enc}_{pk}(x)$ then $\mathrm{dec}_{usk_k}(y) = F(k, x)$.
- Finally, we have a *setup function* that, given a suitable choice of parameters, returns a random pair consisting of a secret key and a public key, generated in accordance with these parameters.
- The set of all these functions defines a protocol of *Functional Encryption*. We assume that they can be computed efficiently.

Applications

- Functional Encryption generalizes several existing (public key) primitives including *Identity-based encryption* (IBE) and *Attribute-based encryption* (ABE).
- An IBE protocol is achieved when, for a functional key k corresponding to users authorized to access any plaintext x, it holds that $F(k, x) = x$. Conversely, if $k = \lambda$ does not grant such access, then $F(\lambda, x) = const$, meaning that no useful decryption is possible for the corresponding users.
- Similarly, in the ABE case, we have $F(k, x) = x$ when k encodes attributes with the necessary permissions to decrypt. Otherwise, it holds that $F(\lambda, x) = const$.
- Standard public key cryptography is obtained for a single functional key k such that $F(k, x) = x$.

Applications

- Functional Encryption generalizes several existing (public key) primitives including *Identity-based encryption* (IBE) and *Attribute-based encryption* (ABE).
- An IBE protocol is achieved when, for a functional key k corresponding to users authorized to access any plaintext x, it holds that $F(k, x) = x$. Conversely, if $k = \lambda$ does not grant such access, then $F(\lambda, x) = const$, meaning that no useful decryption is possible for the corresponding users.
- Similarly, in the ABE case, we have $F(k, x) = x$ when k encodes attributes with the necessary permissions to decrypt. Otherwise, it holds that $F(\lambda, x) = const$.
- Standard public key cryptography is obtained for a single functional key k such that $F(k, x) = x$.

Applications

- Functional Encryption generalizes several existing (public key) primitives including *Identity-based encryption* (IBE) and *Attribute-based encryption* (ABE).
- An IBE protocol is achieved when, for a functional key k corresponding to users authorized to access any plaintext x, it holds that $F(k, x) = x$. Conversely, if $k = \lambda$ does not grant such access, then $F(\lambda, x) = const$, meaning that no useful decryption is possible for the corresponding users.
- Similarly, in the ABE case, we have $F(k, x) = x$ when k encodes attributes with the necessary permissions to decrypt. Otherwise, it holds that $F(\lambda, x) = const$.
- Standard public key cryptography is obtained for a single functional key k such that $F(k, x) = x$.

Applications

- Functional Encryption generalizes several existing (public key) primitives including *Identity-based encryption* (IBE) and *Attribute-based encryption* (ABE).
- An IBE protocol is achieved when, for a functional key k corresponding to users authorized to access any plaintext x, it holds that $F(k, x) = x$. Conversely, if $k = \lambda$ does not grant such access, then $F(\lambda, x) = const$, meaning that no useful decryption is possible for the corresponding users.
- Similarly, in the ABE case, we have $F(k, x) = x$ when k encodes attributes with the necessary permissions to decrypt. Otherwise, it holds that $F(\lambda, x) = const$.
- Standard public key cryptography is obtained for a single functional key k such that $F(k, x) = x$.

Applications

- Functional Encryption generalizes several existing (public key) primitives including *Identity-based encryption* (IBE) and *Attribute-based encryption* (ABE).
- An IBE protocol is achieved when, for a functional key k corresponding to users authorized to access any plaintext x, it holds that $F(k, x) = x$. Conversely, if $k = \lambda$ does not grant such access, then $F(\lambda, x) = const$, meaning that no useful decryption is possible for the corresponding users.
- Similarly, in the ABE case, we have $F(k, x) = x$ when k encodes attributes with the necessary permissions to decrypt. Otherwise, it holds that $F(\lambda, x) = const$.
- Standard public key cryptography is obtained for a single functional key k such that $F(k, x) = x$.

- Aside from these special cases, the main interest in Functional Encryption arises when, as the keys k vary, the functions $x \mapsto F(k, x)$ are relevant for Data Analysis and Machine Learning.
- These functions can be linear (inner product), quadratic, or of higher degree. Applications involving linear functions include linear regression, linear classification. Many neural networks fall into the quadratic case.
- Functional Encryption is hence a tool of choice for *Privacy-Preserving Machine Learning* (or *Data Analysis*) in a context of Cloud Computing.
- Indeed, the functions $x \mapsto F(k, x)$ can be used on a cloud platform by a user holding a functional key k without revealing the plaintext x of another user and without requiring any additional effort from that user or others.

- Aside from these special cases, the main interest in Functional Encryption arises when, as the keys k vary, the functions $x \mapsto F(k, x)$ are relevant for Data Analysis and Machine Learning.
- These functions can be linear (inner product), quadratic, or of higher degree. Applications involving linear functions include linear regression, linear classification. Many neural networks fall into the quadratic case.
- Functional Encryption is hence a tool of choice for *Privacy-Preserving Machine Learning* (or *Data Analysis*) in a context of Cloud Computing.
- Indeed, the functions $x \mapsto F(k, x)$ can be used on a cloud platform by a user holding a functional key k without revealing the plaintext x of another user and without requiring any additional effort from that user or others.

- Aside from these special cases, the main interest in Functional Encryption arises when, as the keys k vary, the functions $x \mapsto F(k, x)$ are relevant for Data Analysis and Machine Learning.
- These functions can be linear (inner product), quadratic, or of higher degree. Applications involving linear functions include linear regression, linear classification. Many neural networks fall into the quadratic case.
- Functional Encryption is hence a tool of choice for *Privacy-Preserving Machine Learning* (or *Data Analysis*) in a context of Cloud Computing.
- Indeed, the functions $x \mapsto F(k, x)$ can be used on a cloud platform by a user holding a functional key k without revealing the plaintext x of another user and without requiring any additional effort from that user or others.

- Aside from these special cases, the main interest in Functional Encryption arises when, as the keys k vary, the functions $x \mapsto F(k, x)$ are relevant for Data Analysis and Machine Learning.
- These functions can be linear (inner product), quadratic, or of higher degree. Applications involving linear functions include linear regression, linear classification. Many neural networks fall into the quadratic case.
- Functional Encryption is hence a tool of choice for *Privacy-Preserving Machine Learning* (or *Data Analysis*) in a context of Cloud Computing.
- Indeed, the functions $x \mapsto F(k, x)$ can be used on a cloud platform by a user holding a functional key k without revealing the plaintext x of another user and without requiring any additional effort from that user or others.

Comparison with Homomorphic Encryption

- Functional Encryption is an interesting alternative to *Homomorphic Encryption*.
- A primary distinction is that in the latter protocol, the functional value $F(x)$ of the plaintext x is only accessible to the owner of a full decryption key, who computes $F(x)$ by decrypting the functional value $F(y)$ of the corresponding ciphertext y.
- The value $F(y)$ can be computed on a cloud platform, and the function F can be any function within the framework of *Fully Homomorphic Encryption*.
- If the owner of the functional value $F(x)$ wants to securely communicate it to another user, an additional cryptographic tool is required. This is not the case with Functional Encryption.
- On the other hand, a Functional Encryption protocol is designed for a specific class of functions $x \mapsto F(k, x)$, and the security concerns are tied to that particular class.

Comparison with Homomorphic Encryption

- Functional Encryption is an interesting alternative to *Homomorphic Encryption*.
- A primary distinction is that in the latter protocol, the functional value $F(x)$ of the plaintext x is only accessible to the owner of a full decryption key, who computes $F(x)$ by decrypting the functional value $F(y)$ of the corresponding ciphertext y.
- The value $F(y)$ can be computed on a cloud platform, and the function F can be any function within the framework of *Fully Homomorphic Encryption*.
- If the owner of the functional value $F(x)$ wants to securely communicate it to another user, an additional cryptographic tool is required. This is not the case with Functional Encryption.
- On the other hand, a Functional Encryption protocol is designed for a specific class of functions $x \mapsto F(k, x)$, and the security concerns are tied to that particular class.

Comparison with Homomorphic Encryption

- Functional Encryption is an interesting alternative to *Homomorphic Encryption*.
- A primary distinction is that in the latter protocol, the functional value $F(x)$ of the plaintext x is only accessible to the owner of a full decryption key, who computes $F(x)$ by decrypting the functional value $F(y)$ of the corresponding ciphertext y.
- The value $F(y)$ can be computed on a cloud platform, and the function F can be any function within the framework of *Fully Homomorphic Encryption*.
- If the owner of the functional value $F(x)$ wants to securely communicate it to another user, an additional cryptographic tool is required. This is not the case with Functional Encryption.
- On the other hand, a Functional Encryption protocol is designed for a specific class of functions $x \mapsto F(k, x)$, and the security concerns are tied to that particular class.

Comparison with Homomorphic Encryption

- Functional Encryption is an interesting alternative to *Homomorphic Encryption*.
- A primary distinction is that in the latter protocol, the functional value $F(x)$ of the plaintext x is only accessible to the owner of a full decryption key, who computes $F(x)$ by decrypting the functional value $F(y)$ of the corresponding ciphertext y.
- The value $F(y)$ can be computed on a cloud platform, and the function F can be any function within the framework of *Fully Homomorphic Encryption*.
- If the owner of the functional value $F(x)$ wants to securely communicate it to another user, an additional cryptographic tool is required. This is not the case with Functional Encryption.
- On the other hand, a Functional Encryption protocol is designed for a specific class of functions $x \mapsto F(k, x)$, and the security concerns are tied to that particular class.

Comparison with Homomorphic Encryption

- Functional Encryption is an interesting alternative to *Homomorphic Encryption*.
- A primary distinction is that in the latter protocol, the functional value $F(x)$ of the plaintext x is only accessible to the owner of a full decryption key, who computes $F(x)$ by decrypting the functional value $F(y)$ of the corresponding ciphertext y.
- The value $F(y)$ can be computed on a cloud platform, and the function F can be any function within the framework of *Fully Homomorphic Encryption*.
- If the owner of the functional value $F(x)$ wants to securely communicate it to another user, an additional cryptographic tool is required. This is not the case with Functional Encryption.
- On the other hand, a Functional Encryption protocol is designed for a specific class of functions $x \mapsto F(k, x)$, and the security concerns are tied to that particular class.

Comparison with Homomorphic Encryption

- Functional Encryption is an interesting alternative to *Homomorphic Encryption*.
- A primary distinction is that in the latter protocol, the functional value $F(x)$ of the plaintext x is only accessible to the owner of a full decryption key, who computes $F(x)$ by decrypting the functional value $F(y)$ of the corresponding ciphertext y.
- The value $F(y)$ can be computed on a cloud platform, and the function F can be any function within the framework of *Fully Homomorphic Encryption*.
- If the owner of the functional value $F(x)$ wants to securely communicate it to another user, an additional cryptographic tool is required. This is not the case with Functional Encryption.
- On the other hand, a Functional Encryption protocol is designed for a specific class of functions $x \mapsto F(k, x)$, and the security concerns are tied to that particular class.

Security

- A first challenge in analyzing the security of Functional Encryption is its inherent vulnerability to ”collusion attacks”, where multiple users may collaborate to share the information available to them.
- Specifically, the decryptors could exchange the knowledge of the functional values $v_i = F(k_i, x)$, in order to determine the plaintext x.
- In other words, a collusion attack corresponds to solving the polynomial system

$$\begin{cases} F(k_1, x) &= v_1 \\ &\vdots \\ F(k_r, x) &= v_r \end{cases}$$

Security

- A first challenge in analyzing the security of Functional Encryption is its inherent vulnerability to ”collusion attacks”, where multiple users may collaborate to share the information available to them.
- Specifically, the decryptors could exchange the knowledge of the functional values $v_i = F(k_i, x)$, in order to determine the plaintext x.
- In other words, a collusion attack corresponds to solving the polynomial system

$$\begin{cases} F(k_1, x) &= v_1 \\ & \vdots \\ F(k_r, x) &= v_r \end{cases}$$

Security

- A first challenge in analyzing the security of Functional Encryption is its inherent vulnerability to ”collusion attacks”, where multiple users may collaborate to share the information available to them.
- Specifically, the decryptors could exchange the knowledge of the functional values $v_i = F(k_i, x)$, in order to determine the plaintext x.
- In other words, a collusion attack corresponds to solving the polynomial system

$$\begin{cases} F(k_1, x) &= v_1 \\ & \vdots \\ F(k_r, x) &= v_r \end{cases}$$

Security

- A first challenge in analyzing the security of Functional Encryption is its inherent vulnerability to ”collusion attacks”, where multiple users may collaborate to share the information available to them.
- Specifically, the decryptors could exchange the knowledge of the functional values $v_i = F(k_i, x)$, in order to determine the plaintext x.
- In other words, a collusion attack corresponds to solving the polynomial system

$$\begin{cases} F(k_1, x) &= v_1 \\ &\vdots \\ F(k_r, x) &= v_r \end{cases}$$

- This type of attack is particularly risky in the case that the functions $x \mapsto F(k_i, x)$ are all linear ones. In this case, the protocol is called an *Inner Product Functional Encryption*.
- A good practice is to maintain a small number of functional keys relative to the plaintext length. This assumption is not overly restrictive in practical applications.
- The commonly used paradigm for performing a security analysis of a Functional Encryption protocol is “simulation-based security”.
- This security proof essentially consists in showing that, except with negligible probability, the real encryption-decryption process is indistinguishable from that of a “simulator”. This agent randomly alters the input plaintexts without affecting the decryption output, namely the set of functional values obtained from a collusion attack.

- This type of attack is particularly risky in the case that the functions $x \mapsto F(k_i, x)$ are all linear ones. In this case, the protocol is called an *Inner Product Functional Encryption*.
- A good practice is to maintain a small number of functional keys relative to the plaintext length. This assumption is not overly restrictive in practical applications.
- The commonly used paradigm for performing a security analysis of a Functional Encryption protocol is “simulation-based security”.
- This security proof essentially consists in showing that, except with negligible probability, the real encryption-decryption process is indistinguishable from that of a “simulator”. This agent randomly alters the input plaintexts without affecting the decryption output, namely the set of functional values obtained from a collusion attack.

- This type of attack is particularly risky in the case that the functions $x \mapsto F(k_i, x)$ are all linear ones. In this case, the protocol is called an *Inner Product Functional Encryption*.
- A good practice is to maintain a small number of functional keys relative to the plaintext length. This assumption is not overly restrictive in practical applications.
- The commonly used paradigm for performing a security analysis of a Functional Encryption protocol is “simulation-based security”.
- This security proof essentially consists in showing that, except with negligible probability, the real encryption-decryption process is indistinguishable from that of a “simulator”. This agent randomly alters the input plaintexts without affecting the decryption output, namely the set of functional values obtained from a collusion attack.

- This type of attack is particularly risky in the case that the functions $x \mapsto F(k_i, x)$ are all linear ones. In this case, the protocol is called an *Inner Product Functional Encryption*.
- A good practice is to maintain a small number of functional keys relative to the plaintext length. This assumption is not overly restrictive in practical applications.
- The commonly used paradigm for performing a security analysis of a Functional Encryption protocol is “simulation-based security”.
- This security proof essentially consists in showing that, except with negligible probability, the real encryption-decryption process is indistinguishable from that of a “simulator”. This agent randomly alters the input plaintexts without affecting the decryption output, namely the set of functional values obtained from a collusion attack.

Inner Product Functional Encryption

- We now present an IPFE protocol based on multivariate cryptography, requiring the introduction of some new notation.
- Let $\mathbb{F}$ be a finite field and let $\mathbb{F}^d$ denote a vector space over $\mathbb{F}$ of dimension d. In an IPFE protocol, the set K of functional keys and the set X of plaintexts both correspond to $\mathbb{F}^d$, while the set V of functional values coincides with $\mathbb{F}$.
- Moreover, for any functional key $a = (a_1, \ldots, a_d)$ and each plaintext $b = (b_1, \ldots, b_d)$, we have by definition

$$F(a, b) = \langle a, b \rangle = \sum_i a_i b_i.$$

Inner Product Functional Encryption

- We now present an IPFE protocol based on multivariate cryptography, requiring the introduction of some new notation.
- Let $\mathbb{F}$ be a finite field and let $\mathbb{F}^d$ denote a vector space over $\mathbb{F}$ of dimension d. In an IPFE protocol, the set K of functional keys and the set X of plaintexts both correspond to $\mathbb{F}^d$, while the set V of functional values coincides with $\mathbb{F}$.
- Moreover, for any functional key $a = (a_1, \ldots, a_d)$ and each plaintext $b = (b_1, \ldots, b_d)$, we have by definition

$$F(a, b) = \langle a, b \rangle = \sum_i a_i b_i.$$

Inner Product Functional Encryption

- We now present an IPFE protocol based on multivariate cryptography, requiring the introduction of some new notation.
- Let $\mathbb{F}$ be a finite field and let $\mathbb{F}^d$ denote a vector space over $\mathbb{F}$ of dimension d. In an IPFE protocol, the set K of functional keys and the set X of plaintexts both correspond to $\mathbb{F}^d$, while the set V of functional values coincides with $\mathbb{F}$.
- Moreover, for any functional key $a = (a_1, \ldots, a_d)$ and each plaintext $b = (b_1, \ldots, b_d)$, we have by definition

$$F(a, b) = \langle a, b \rangle = \sum_i a_i b_i.$$

Inner Product Functional Encryption

- We now present an IPFE protocol based on multivariate cryptography, requiring the introduction of some new notation.
- Let $\mathbb{F}$ be a finite field and let $\mathbb{F}^d$ denote a vector space over $\mathbb{F}$ of dimension d. In an IPFE protocol, the set K of functional keys and the set X of plaintexts both correspond to $\mathbb{F}^d$, while the set V of functional values coincides with $\mathbb{F}$.
- Moreover, for any functional key $a = (a_1, \ldots, a_d)$ and each plaintext $b = (b_1, \ldots, b_d)$, we have by definition

$$F(a, b) = \langle a, b \rangle = \sum_i a_i b_i.$$

Oil-Vinegar maps

- Let $S = \mathbb{F}[x_1, \ldots, x_n]$ be the algebra of the polynomials with coefficients in the field $\mathbb{F}$ and variables $x_1, \ldots, x_n$.
- For all $1 \leq k \leq m$, let $f_k \in S$ be a *quadratic form*, that is, a homogeneous polynomial of degree 2.
- Put $F = (f_1, \ldots, f_m) \in S^m$. By abuse of notation, we also denote by $F : \mathbb{F}^n \to \mathbb{F}^m$ the *quadratic map* such that, for all vectors $v = (v_1, \ldots, v_n) \in \mathbb{F}^n$

$$F(v) = (f_1(v), \ldots, f_m(v)).$$

- We call F an *oil-vinegar map* if there exists a vector subspace $0 \neq O \subset \mathbb{F}^n$ such that $F(O) = 0$. We call O the *oil subspace* of F.

Oil-Vinegar maps

- Let $S = \mathbb{F}[x_1, \ldots, x_n]$ be the algebra of the polynomials with coefficients in the field $\mathbb{F}$ and variables $x_1, \ldots, x_n$.
- For all $1 \leq k \leq m$, let $f_k \in S$ be a *quadratic form*, that is, a homogeneous polynomial of degree 2.
- Put $F = (f_1, \ldots, f_m) \in S^m$. By abuse of notation, we also denote by $F : \mathbb{F}^n \to \mathbb{F}^m$ the *quadratic map* such that, for all vectors $v = (v_1, \ldots, v_n) \in \mathbb{F}^n$

$$F(v) = (f_1(v), \ldots, f_m(v)).$$

- We call F an *oil-vinegar map* if there exists a vector subspace $0 \neq O \subset \mathbb{F}^n$ such that $F(O) = 0$. We call O the *oil subspace* of F.

Oil-Vinegar maps

- Let $S = \mathbb{F}[x_1, \ldots, x_n]$ be the algebra of the polynomials with coefficients in the field $\mathbb{F}$ and variables $x_1, \ldots, x_n$.
- For all $1 \leq k \leq m$, let $f_k \in S$ be a *quadratic form*, that is, a homogeneous polynomial of degree 2.
- Put $F = (f_1, \ldots, f_m) \in S^m$. By abuse of notation, we also denote by $F : \mathbb{F}^n \to \mathbb{F}^m$ the *quadratic map* such that, for all vectors $v = (v_1, \ldots, v_n) \in \mathbb{F}^n$

$$F(v) = (f_1(v), \ldots, f_m(v)).$$

- We call F an *oil-vinegar map* if there exists a vector subspace $0 \neq O \subset \mathbb{F}^n$ such that $F(O) = 0$. We call O the *oil subspace* of F.

Oil-Vinegar maps

- Let $S = \mathbb{F}[x_1, \ldots, x_n]$ be the algebra of the polynomials with coefficients in the field $\mathbb{F}$ and variables $x_1, \ldots, x_n$.
- For all $1 \leq k \leq m$, let $f_k \in S$ be a *quadratic form*, that is, a homogeneous polynomial of degree 2.
- Put $F = (f_1, \ldots, f_m) \in S^m$. By abuse of notation, we also denote by $F : \mathbb{F}^n \to \mathbb{F}^m$ the *quadratic map* such that, for all vectors $v = (v_1, \ldots, v_n) \in \mathbb{F}^n$

$$F(v) = (f_1(v), \ldots, f_m(v)).$$

- We call F an *oil-vinegar map* if there exists a vector subspace $0 \neq O \subset \mathbb{F}^n$ such that $F(O) = 0$. We call O the *oil subspace* of F.

Oil-Vinegar maps

- Let $S = \mathbb{F}[x_1, \ldots, x_n]$ be the algebra of the polynomials with coefficients in the field $\mathbb{F}$ and variables $x_1, \ldots, x_n$.
- For all $1 \leq k \leq m$, let $f_k \in S$ be a *quadratic form*, that is, a homogeneous polynomial of degree 2.
- Put $F = (f_1, \ldots, f_m) \in S^m$. By abuse of notation, we also denote by $F : \mathbb{F}^n \to \mathbb{F}^m$ the *quadratic map* such that, for all vectors $v = (v_1, \ldots, v_n) \in \mathbb{F}^n$

$$F(v) = (f_1(v), \ldots, f_m(v)).$$

- We call F an *oil-vinegar map* if there exists a vector subspace $0 \neq O \subset \mathbb{F}^n$ such that $F(O) = 0$. We call O the *oil subspace* of F.

Preimages of oil-vinegar maps

- By the knowledge of the oil subspace $O \subset \mathbb{F}^n$, it is easy to compute a preimage $v \in F^{-1}(w) \subset \mathbb{F}^n$ of a vector $w \in \mathbb{F}^m$ with respect to an oil-vinegar map $F : \mathbb{F}^n \to \mathbb{F}^m$.
- Let $\bar{S} = \mathbb{F}[x_1, \ldots, x_n, y_1, \ldots, y_n]$ be the polynomial algebra in the two disjoint sets of variables $\{x_1, \ldots, x_n\}$ and $\{y_1, \ldots, y_n\}$.
- For all $1 \leq k \leq m$, let $\bar{f}_k \in \bar{S}$ be the polar form of the quadratic form f_k, namely

$$\bar{f}_k(x, y) = f_k(x + y) - f_k(x) - f_k(y).$$

- Let $\bar{F} = (\bar{f}_1, \ldots, \bar{f}_m) \in \bar{S}^m$. By abuse of notation, we denote by $\bar{F} : \mathbb{F}^n \times \mathbb{F}^n \to \mathbb{F}^m$ the bilinear map such that, for all $u, v \in \mathbb{F}^n$

$$\bar{F}(u, v) = (\bar{f}_1(u, v), \ldots, \bar{f}_m(u, v)).$$

We call $\bar{F}$ the *polar map* of the quadratic map F.

Preimages of oil-vinegar maps

- By the knowledge of the oil subspace $O \subset \mathbb{F}^n$, it is easy to compute a preimage $v \in F^{-1}(w) \subset \mathbb{F}^n$ of a vector $w \in \mathbb{F}^m$ with respect to an oil-vinegar map $F : \mathbb{F}^n \to \mathbb{F}^m$.
- Let $\bar{S} = \mathbb{F}[x_1, \ldots, x_n, y_1, \ldots, y_n]$ be the polynomial algebra in the two disjoint sets of variables $\{x_1, \ldots, x_n\}$ and $\{y_1, \ldots, y_n\}$.
- For all $1 \leq k \leq m$, let $\bar{f}_k \in \bar{S}$ be the polar form of the quadratic form f_k, namely

$$\bar{f}_k(x, y) = f_k(x + y) - f_k(x) - f_k(y).$$

- Let $\bar{F} = (\bar{f}_1, \ldots, \bar{f}_m) \in \bar{S}^m$. By abuse of notation, we denote by $\bar{F} : \mathbb{F}^n \times \mathbb{F}^n \to \mathbb{F}^m$ the bilinear map such that, for all $u, v \in \mathbb{F}^n$

$$\bar{F}(u, v) = (\bar{f}_1(u, v), \ldots, \bar{f}_m(u, v)).$$

We call $\bar{F}$ the *polar map* of the quadratic map F.

Preimages of oil-vinegar maps

- By the knowledge of the oil subspace $O \subset \mathbb{F}^n$, it is easy to compute a preimage $v \in F^{-1}(w) \subset \mathbb{F}^n$ of a vector $w \in \mathbb{F}^m$ with respect to an oil-vinegar map $F : \mathbb{F}^n \to \mathbb{F}^m$.
- Let $\bar{S} = \mathbb{F}[x_1, \ldots, x_n, y_1, \ldots, y_n]$ be the polynomial algebra in the two disjoint sets of variables $\{x_1, \ldots, x_n\}$ and $\{y_1, \ldots, y_n\}$.
- For all $1 \leq k \leq m$, let $\bar{f}_k \in \bar{S}$ be the polar form of the quadratic form f_k, namely

$$\bar{f}_k(x, y) = f_k(x + y) - f_k(x) - f_k(y).$$

- Let $\bar{F} = (\bar{f}_1, \ldots, \bar{f}_m) \in \bar{S}^m$. By abuse of notation, we denote by $\bar{F} : \mathbb{F}^n \times \mathbb{F}^n \to \mathbb{F}^m$ the bilinear map such that, for all $u, v \in \mathbb{F}^n$

$$\bar{F}(u, v) = (\bar{f}_1(u, v), \ldots, \bar{f}_m(u, v)).$$

We call $\bar{F}$ the *polar map* of the quadratic map F.

Preimages of oil-vinegar maps

- By the knowledge of the oil subspace $O \subset \mathbb{F}^n$, it is easy to compute a preimage $v \in F^{-1}(w) \subset \mathbb{F}^n$ of a vector $w \in \mathbb{F}^m$ with respect to an oil-vinegar map $F : \mathbb{F}^n \to \mathbb{F}^m$.
- Let $\bar{S} = \mathbb{F}[x_1, \ldots, x_n, y_1, \ldots, y_n]$ be the polynomial algebra in the two disjoint sets of variables $\{x_1, \ldots, x_n\}$ and $\{y_1, \ldots, y_n\}$.
- For all $1 \leq k \leq m$, let $\bar{f}_k \in \bar{S}$ be the polar form of the quadratic form f_k, namely

$$\bar{f}_k(x, y) = f_k(x + y) - f_k(x) - f_k(y).$$

- Let $\bar{F} = (\bar{f}_1, \ldots, \bar{f}_m) \in \bar{S}^m$. By abuse of notation, we denote by $\bar{F} : \mathbb{F}^n \times \mathbb{F}^n \to \mathbb{F}^m$ the bilinear map such that, for all $u, v \in \mathbb{F}^n$

$$\bar{F}(u, v) = (\bar{f}_1(u, v), \ldots, \bar{f}_m(u, v)).$$

We call $\bar{F}$ the *polar map* of the quadratic map F.

Preimages of oil-vinegar maps

- By the knowledge of the oil subspace $O \subset \mathbb{F}^n$, it is easy to compute a preimage $v \in F^{-1}(w) \subset \mathbb{F}^n$ of a vector $w \in \mathbb{F}^m$ with respect to an oil-vinegar map $F : \mathbb{F}^n \to \mathbb{F}^m$.
- Let $\bar{S} = \mathbb{F}[x_1, \ldots, x_n, y_1, \ldots, y_n]$ be the polynomial algebra in the two disjoint sets of variables $\{x_1, \ldots, x_n\}$ and $\{y_1, \ldots, y_n\}$.
- For all $1 \leq k \leq m$, let $\bar{f}_k \in \bar{S}$ be the polar form of the quadratic form f_k, namely

$$\bar{f}_k(x, y) = f_k(x + y) - f_k(x) - f_k(y).$$

- Let $\bar{F} = (\bar{f}_1, \ldots, \bar{f}_m) \in \bar{S}^m$. By abuse of notation, we denote by $\bar{F} : \mathbb{F}^n \times \mathbb{F}^n \to \mathbb{F}^m$ the bilinear map such that, for all $u, v \in \mathbb{F}^n$

$$\bar{F}(u, v) = (\bar{f}_1(u, v), \ldots, \bar{f}_m(u, v)).$$

We call $\bar{F}$ the *polar map* of the quadratic map F.

- Assume F is an oil-vinegar map with oil subspace $O \subset \mathbb{F}^n$ and put $r = \dim_{\mathbb{F}} O$.
- Fix a random vector $u \in \mathbb{F}^n$. If $o \in O$ and hence $F(o) = 0$, then

$$w = F(u + o) = F(u) + F(o) + \bar{F}(u, o) = F(u) + \bar{F}(u, o).$$

- For computing a preimage $v = u + o \in F^{-1}(w)$ it is sufficient therefore to determine a suitable vector $o \in O$. This vector is a solution of the following system of linear equations

$$\begin{cases} l_1(x) & = & 0, \\ & \vdots & \\ l_{n-r}(x) & = & 0, \\ \bar{f}_1(u, x) & = & w_1 - f_1(u), \\ & \vdots & \\ \bar{f}_m(u, x) & = & w_m - f_m(u). \end{cases}$$

where $\{l_1, \ldots, l_{n-r}\}$ are the linear forms defining the subspace O.

- Assume F is an oil-vinegar map with oil subspace $O \subset \mathbb{F}^n$ and put $r = \dim_{\mathbb{F}} O$.
- Fix a random vector $u \in \mathbb{F}^n$. If $o \in O$ and hence $F(o) = 0$, then

$$w = F(u + o) = F(u) + F(o) + \bar{F}(u, o) = F(u) + \bar{F}(u, o).$$

- For computing a preimage $v = u + o \in F^{-1}(w)$ it is sufficient therefore to determine a suitable vector $o \in O$. This vector is a solution of the following system of linear equations

$$\begin{cases} l_1(x) & = & 0, \\ & \vdots & \\ l_{n-r}(x) & = & 0, \\ \bar{f}_1(u, x) & = & w_1 - f_1(u), \\ & \vdots & \\ \bar{f}_m(u, x) & = & w_m - f_m(u). \end{cases}$$

where $\{l_1, \ldots, l_{n-r}\}$ are the linear forms defining the subspace O.

- Assume F is an oil-vinegar map with oil subspace $O \subset \mathbb{F}^n$ and put $r = \dim_{\mathbb{F}} O$.
- Fix a random vector $u \in \mathbb{F}^n$. If $o \in O$ and hence $F(o) = 0$, then

$$w = F(u + o) = F(u) + F(o) + \bar{F}(u, o) = F(u) + \bar{F}(u, o).$$

- For computing a preimage $v = u + o \in F^{-1}(w)$ it is sufficient therefore to determine a suitable vector $o \in O$. This vector is a solution of the following system of linear equations

$$\begin{cases} l_1(x) & = & 0, \\ & \vdots & \\ l_{n-r}(x) & = & 0, \\ \bar{f}_1(u, x) & = & w_1 - f_1(u), \\ & \vdots & \\ \bar{f}_m(u, x) & = & w_m - f_m(u). \end{cases}$$

where $\{l_1, \ldots, l_{n-r}\}$ are the linear forms defining the subspace O.

- Typically $m = r$, resulting in a square linear system. If the system is singular (with probability $1/q$ where $q = |\mathbb{F}|$), it suffices to select a different random vector $u \in \mathbb{F}^n$.
- Without knowledge of the oil subspace O, computing a preimage is equivalent to solving a system of quadratic equations over a finite field, a problem that is well-known to be NP-hard.
- These are the fundamental concepts underlying the modern description of the digital signature protocol known as *Unbalanced Oil-Vinegar*. In this description, the public key is an oil-vinegar map F, while the secret key consists of F along with its oil subspace O. A signature is simply a preimage with respect to F.
- For security reasons (Knipis-Shamir attack), it is assumed that $n > 2m$ and $\dim_{\mathbb{F}} O = m$.

- Typically $m = r$, resulting in a square linear system. If the system is singular (with probability $1/q$ where $q = |\mathbb{F}|$), it suffices to select a different random vector $u \in \mathbb{F}^n$.
- Without knowledge of the oil subspace O, computing a preimage is equivalent to solving a system of quadratic equations over a finite field, a problem that is well-known to be NP-hard.
- These are the fundamental concepts underlying the modern description of the digital signature protocol known as *Unbalanced Oil-Vinegar*. In this description, the public key is an oil-vinegar map F, while the secret key consists of F along with its oil subspace O. A signature is simply a preimage with respect to F.
- For security reasons (Knipis-Shamir attack), it is assumed that $n > 2m$ and $\dim_{\mathbb{F}} O = m$.

- Typically $m = r$, resulting in a square linear system. If the system is singular (with probability $1/q$ where $q = |\mathbb{F}|$), it suffices to select a different random vector $u \in \mathbb{F}^n$.
- Without knowledge of the oil subspace O, computing a preimage is equivalent to solving a system of quadratic equations over a finite field, a problem that is well-known to be NP-hard.
- These are the fundamental concepts underlying the modern description of the digital signature protocol known as *Unbalanced Oil-Vinegar*. In this description, the public key is an oil-vinegar map F, while the secret key consists of F along with its oil subspace O. A signature is simply a preimage with respect to F.
- For security reasons (Knipis-Shamir attack), it is assumed that $n > 2m$ and $\dim_{\mathbb{F}} O = m$.

- Typically $m = r$, resulting in a square linear system. If the system is singular (with probability $1/q$ where $q = |\mathbb{F}|$), it suffices to select a different random vector $u \in \mathbb{F}^n$.
- Without knowledge of the oil subspace O, computing a preimage is equivalent to solving a system of quadratic equations over a finite field, a problem that is well-known to be NP-hard.
- These are the fundamental concepts underlying the modern description of the digital signature protocol known as *Unbalanced Oil-Vinegar*. In this description, the public key is an oil-vinegar map F, while the secret key consists of F along with its oil subspace O. A signature is simply a preimage with respect to F.
- For security reasons (Knipis-Shamir attack), it is assumed that $n > 2m$ and $\dim_{\mathbb{F}} O = m$.

IPFE based on UOV

- In 2021, Debnath, Mesnager, Dey and Kundu proposed a protocol for IPFE based on quadratic maps. Here, we present a variant of this protocol involving modern UOV description.

- Let $\{t_1, \ldots, t_d\}$ and $\{x_1, \ldots, x_n\}$ be two disjoint sets of variables and consider the polynomial algebras $R = \mathbb{F}[t_1, \ldots, t_d]$ and $S = \mathbb{F}[x_1, \ldots, x_n]$. We put

$$P = R[x_1, \ldots, x_n] = \mathbb{F}[t_1, \ldots, t_d, x_1, \ldots, x_n].$$

- Consider m quadratic forms in the variables $x_1, \ldots, x_n$ whose coefficients are polynomials in the variables $t_1, \ldots, t_d$. Precisely, if $t = (t_1, \ldots, t_d) \in R^d$ and $x = (x_1, \ldots, x_n) \in S^n$, for each $1 \leq k \leq m$ we define

$$f_k^{(t)}(x) = x\, A(t)_k\, x^T \in P$$

where $A(t)_k$ is an upper triangular matrix with entries in R.

IPFE based on UOV

- In 2021, Debnath, Mesnager, Dey and Kundu proposed a protocol for IPFE based on quadratic maps. Here, we present a variant of this protocol involving modern UOV description.

- Let $\{t_1, \ldots, t_d\}$ and $\{x_1, \ldots, x_n\}$ be two disjoint sets of variables and consider the polynomial algebras $R = \mathbb{F}[t_1, \ldots, t_d]$ and $S = \mathbb{F}[x_1, \ldots, x_n]$. We put

$$P = R[x_1, \ldots, x_n] = \mathbb{F}[t_1, \ldots, t_d, x_1, \ldots, x_n].$$

- Consider m quadratic forms in the variables $x_1, \ldots, x_n$ whose coefficients are polynomials in the variables $t_1, \ldots, t_d$. Precisely, if $t = (t_1, \ldots, t_d) \in R^d$ and $x = (x_1, \ldots, x_n) \in S^n$, for each $1 \leq k \leq m$ we define

$$f_k^{(t)}(x) = x\, A(t)_k\, x^T \in P$$

where $A(t)_k$ is an upper triangular matrix with entries in R.

IPFE based on UOV

- In 2021, Debnath, Mesnager, Dey and Kundu proposed a protocol for IPFE based on quadratic maps. Here, we present a variant of this protocol involving modern UOV description.

- Let $\{t_1, \ldots, t_d\}$ and $\{x_1, \ldots, x_n\}$ be two disjoint sets of variables and consider the polynomial algebras $R = \mathbb{F}[t_1, \ldots, t_d]$ and $S = \mathbb{F}[x_1, \ldots, x_n]$. We put

$$P = R[x_1, \ldots, x_n] = \mathbb{F}[t_1, \ldots, t_d, x_1, \ldots, x_n].$$

- Consider m quadratic forms in the variables $x_1, \ldots, x_n$ whose coefficients are polynomials in the variables $t_1, \ldots, t_d$. Precisely, if $t = (t_1, \ldots, t_d) \in R^d$ and $x = (x_1, \ldots, x_n) \in S^n$, for each $1 \leq k \leq m$ we define

$$f_k^{(t)}(x) = x\, A(t)_k\, x^T \in P$$

where $A(t)_k$ is an upper triangular matrix with entries in R.

IPFE based on UOV

- In 2021, Debnath, Mesnager, Dey and Kundu proposed a protocol for IPFE based on quadratic maps. Here, we present a variant of this protocol involving modern UOV description.

- Let $\{t_1, \ldots, t_d\}$ and $\{x_1, \ldots, x_n\}$ be two disjoint sets of variables and consider the polynomial algebras $R = \mathbb{F}[t_1, \ldots, t_d]$ and $S = \mathbb{F}[x_1, \ldots, x_n]$. We put

$$P = R[x_1, \ldots, x_n] = \mathbb{F}[t_1, \ldots, t_d, x_1, \ldots, x_n].$$

- Consider m quadratic forms in the variables $x_1, \ldots, x_n$ whose coefficients are polynomials in the variables $t_1, \ldots, t_d$. Precisely, if $t = (t_1, \ldots, t_d) \in R^d$ and $x = (x_1, \ldots, x_n) \in S^n$, for each $1 \leq k \leq m$ we define

$$f_k^{(t)}(x) = x\, A(t)_k\, x^T \in P$$

where $A(t)_k$ is an upper triangular matrix with entries in R.

- Put $F^{(t)} = (f_1^{(t)}, \ldots, f_m^{(t)}) \in P^m$.
- By evaluating the variable vector $t = (t_1, \ldots, t_d)$ at a vector $a = (a_1, \ldots, a_d) \in \mathbb{F}^d$, we obtain the evaluation of $F^{(t)}$ at a quadratic map $F^{(a)} = (f_1^{(a)}, \ldots, f_m^{(a)}) \in S^m$.
- We call $F^{(t)}$ an *oil-vinegar map* if $F^{(a)}$ is an oil-vinegar map, for each $a = (a_1, \ldots, a_d) \in \mathbb{F}^d$. If $O(a) \subset \mathbb{F}^n$ is the oil subspace of $F^{(a)}$, we assume that $\dim_{\mathbb{F}} O(a) = m$, for all $a \in \mathbb{F}^d$.
- Such a map can be easily constructed by means of block upper triangular matrices of type

$$A(t)_k = \begin{pmatrix} A(t)'_k & A(t)''_k \\ 0 & A(t)'''_k \end{pmatrix}$$

where $A(t)'_k$ is an upper triangular $(n-m) \times (n-m)$ matrix, $A(t)''_k$ is an $(n-m) \times m$ matrix and $A(t)'_k$ is an upper triangular $m \times m$ matrix, all with entries in the algebra R.

- Put $F^{(t)} = (f_1^{(t)}, \ldots, f_m^{(t)}) \in P^m$.
- By evaluating the variable vector $t = (t_1, \ldots, t_d)$ at a vector $a = (a_1, \ldots, a_d) \in \mathbb{F}^d$, we obtain the evaluation of $F^{(t)}$ at a quadratic map $F^{(a)} = (f_1^{(a)}, \ldots, f_m^{(a)}) \in S^m$.
- We call $F^{(t)}$ an *oil-vinegar map* if $F^{(a)}$ is an oil-vinegar map, for each $a = (a_1, \ldots, a_d) \in \mathbb{F}^d$. If $O(a) \subset \mathbb{F}^n$ is the oil subspace of $F^{(a)}$, we assume that $\dim_{\mathbb{F}} O(a) = m$, for all $a \in \mathbb{F}^d$.
- Such a map can be easily constructed by means of block upper triangular matrices of type

$$A(t)_k = \begin{pmatrix} A(t)'_k & A(t)''_k \\ 0 & A(t)'''_k \end{pmatrix}$$

where $A(t)'_k$ is an upper triangular $(n-m) \times (n-m)$ matrix, $A(t)''_k$ is an $(n-m) \times m$ matrix and $A(t)'_k$ is an upper triangular $m \times m$ matrix, all with entries in the algebra R.

- Put $F^{(t)} = (f_1^{(t)}, \ldots, f_m^{(t)}) \in P^m$.
- By evaluating the variable vector $t = (t_1, \ldots, t_d)$ at a vector $a = (a_1, \ldots, a_d) \in \mathbb{F}^d$, we obtain the evaluation of $F^{(t)}$ at a quadratic map $F^{(a)} = (f_1^{(a)}, \ldots, f_m^{(a)}) \in S^m$.
- We call $F^{(t)}$ an *oil-vinegar map* if $F^{(a)}$ is an oil-vinegar map, for each $a = (a_1, \ldots, a_d) \in \mathbb{F}^d$. If $O(a) \subset \mathbb{F}^n$ is the oil subspace of $F^{(a)}$, we assume that $\dim_{\mathbb{F}} O(a) = m$, for all $a \in \mathbb{F}^d$.
- Such a map can be easily constructed by means of block upper triangular matrices of type

$$A(t)_k = \begin{pmatrix} A(t)'_k & A(t)''_k \\ 0 & A(t)'''_k \end{pmatrix}$$

where $A(t)'_k$ is an upper triangular $(n-m) \times (n-m)$ matrix, $A(t)''_k$ is an $(n-m) \times m$ matrix and $A(t)'_k$ is an upper triangular $m \times m$ matrix, all with entries in the algebra R.

- Put $F^{(t)} = (f_1^{(t)}, \ldots, f_m^{(t)}) \in P^m$.
- By evaluating the variable vector $t = (t_1, \ldots, t_d)$ at a vector $a = (a_1, \ldots, a_d) \in \mathbb{F}^d$, we obtain the evaluation of $F^{(t)}$ at a quadratic map $F^{(a)} = (f_1^{(a)}, \ldots, f_m^{(a)}) \in S^m$.
- We call $F^{(t)}$ an *oil-vinegar map* if $F^{(a)}$ is an oil-vinegar map, for each $a = (a_1, \ldots, a_d) \in \mathbb{F}^d$. If $O(a) \subset \mathbb{F}^n$ is the oil subspace of $F^{(a)}$, we assume that $\dim_{\mathbb{F}} O(a) = m$, for all $a \in \mathbb{F}^d$.
- Such a map can be easily constructed by means of block upper triangular matrices of type

$$A(t)_k = \begin{pmatrix} A(t)'_k & A(t)''_k \\ 0 & A(t)'''_k \end{pmatrix}$$

where $A(t)'_k$ is an upper triangular $(n-m) \times (n-m)$ matrix, $A(t)''_k$ is an $(n-m) \times m$ matrix and $A(t)'_k$ is an upper triangular $m \times m$ matrix, all with entries in the algebra R.

Secret and public keys

- Denote by $O(t) \subset R^n$ the vector subspace such that $O(a) \subset \mathbb{F}^n$ is the oil subspace of $F^{(a)}$, for all $a \in \mathbb{F}^d$.
- In this IPFE protocol, the secret key is given by the pair $(F^{(t)}, O(t))$, the public key by the map $F^{(t)}$ alone, and the user secret key by the pair $(F^{(a)}, O(a))$, for all functional keys $a \in \mathbb{F}^d$.
- For simplicity, we refer to the public key owner as Alice and the user secret key owner as Bob.
- We recall that the goal of the functional decryption is to determine the inner product $\langle a, b \rangle = \sum_i a_i b_i$ where $b = (b_1, \ldots, b_d)$ is Alice's plaintext and $a = (a_1, \ldots, a_d)$ is Bob's functional key.

Secret and public keys

- Denote by $O(t) \subset R^n$ the vector subspace such that $O(a) \subset \mathbb{F}^n$ is the oil subspace of $F^{(a)}$, for all $a \in \mathbb{F}^d$.
- In this IPFE protocol, the secret key is given by the pair $(F^{(t)}, O(t))$, the public key by the map $F^{(t)}$ alone, and the user secret key by the pair $(F^{(a)}, O(a))$, for all functional keys $a \in \mathbb{F}^d$.
- For simplicity, we refer to the public key owner as Alice and the user secret key owner as Bob.
- We recall that the goal of the functional decryption is to determine the inner product $\langle a, b \rangle = \sum_i a_i b_i$ where $b = (b_1, \ldots, b_d)$ is Alice's plaintext and $a = (a_1, \ldots, a_d)$ is Bob's functional key.

Secret and public keys

- Denote by $O(t) \subset R^n$ the vector subspace such that $O(a) \subset \mathbb{F}^n$ is the oil subspace of $F^{(a)}$, for all $a \in \mathbb{F}^d$.
- In this IPFE protocol, the secret key is given by the pair $(F^{(t)}, O(t))$, the public key by the map $F^{(t)}$ alone, and the user secret key by the pair $(F^{(a)}, O(a))$, for all functional keys $a \in \mathbb{F}^d$.
- For simplicity, we refer to the public key owner as Alice and the user secret key owner as Bob.
- We recall that the goal of the functional decryption is to determine the inner product $\langle a, b \rangle = \sum_i a_i b_i$ where $b = (b_1, \ldots, b_d)$ is Alice's plaintext and $a = (a_1, \ldots, a_d)$ is Bob's functional key.

Secret and public keys

- Denote by $O(t) \subset R^n$ the vector subspace such that $O(a) \subset \mathbb{F}^n$ is the oil subspace of $F^{(a)}$, for all $a \in \mathbb{F}^d$.
- In this IPFE protocol, the secret key is given by the pair $(F^{(t)}, O(t))$, the public key by the map $F^{(t)}$ alone, and the user secret key by the pair $(F^{(a)}, O(a))$, for all functional keys $a \in \mathbb{F}^d$.
- For simplicity, we refer to the public key owner as Alice and the user secret key owner as Bob.
- We recall that the goal of the functional decryption is to determine the inner product $\langle a, b \rangle = \sum_i a_i b_i$ where $b = (b_1, \ldots, b_d)$ is Alice's plaintext and $a = (a_1, \ldots, a_d)$ is Bob's functional key.

Secret and public keys

- Denote by $O(t) \subset R^n$ the vector subspace such that $O(a) \subset \mathbb{F}^n$ is the oil subspace of $F^{(a)}$, for all $a \in \mathbb{F}^d$.
- In this IPFE protocol, the secret key is given by the pair $(F^{(t)}, O(t))$, the public key by the map $F^{(t)}$ alone, and the user secret key by the pair $(F^{(a)}, O(a))$, for all functional keys $a \in \mathbb{F}^d$.
- For simplicity, we refer to the public key owner as Alice and the user secret key owner as Bob.
- We recall that the goal of the functional decryption is to determine the inner product $\langle a, b \rangle = \sum_i a_i b_i$ where $b = (b_1, \ldots, b_d)$ is Alice's plaintext and $a = (a_1, \ldots, a_d)$ is Bob's functional key.

Functional encryption

- Alice computes the linear form $\langle t, b \rangle = \sum_i t_i b_i \in R$.
- A vector $u(t) = (u(t)_1, \ldots, u(t)_n) \in R^n$ is defined such that each $u(t)_i$ is a linear form in the variables $t_1, \ldots, t_d$ and $\sum_i u(t)_i = \langle t, b \rangle$. Indeed, it is sufficient to define $u(t)_n = \langle t, b \rangle - \sum_{i=1}^{n-1} u(t)_i$, while the other entries of $u(t)$ can be chosen arbitrarily.
- Alice also computes a control vector $v(t) = u(t) + c$, where by definition $c = (c_1, \ldots, c_n) \in \mathbb{F}^n$ is a randomly chosen vector.
- Finally, she computes the values $\bar{u}(t) = F^{(t)}(u(t))$ and $\bar{v}(t) = F^{(t)}(v(t))$.
- The ciphertext computed by Alice for the plaintext $b = (b_1, \ldots, b_d)$ is defined as the triplet $\mathrm{CT}^{(t)}(b) = (\bar{u}(t), \bar{v}(t), c)$.

Functional encryption

- Alice computes the linear form $\langle t, b \rangle = \sum_i t_i b_i \in R$.
- A vector $u(t) = (u(t)_1, \ldots, u(t)_n) \in R^n$ is defined such that each $u(t)_i$ is a linear form in the variables $t_1, \ldots, t_d$ and $\sum_i u(t)_i = \langle t, b \rangle$. Indeed, it is sufficient to define $u(t)_n = \langle t, b \rangle - \sum_{i=1}^{n-1} u(t)_i$, while the other entries of $u(t)$ can be chosen arbitrarily.
- Alice also computes a control vector $v(t) = u(t) + c$, where by definition $c = (c_1, \ldots, c_n) \in \mathbb{F}^n$ is a randomly chosen vector.
- Finally, she computes the values $\bar{u}(t) = F^{(t)}(u(t))$ and $\bar{v}(t) = F^{(t)}(v(t))$.
- The ciphertext computed by Alice for the plaintext $b = (b_1, \ldots, b_d)$ is defined as the triplet $\mathrm{CT}^{(t)}(b) = (\bar{u}(t), \bar{v}(t), c)$.

Functional encryption

- Alice computes the linear form $\langle t, b \rangle = \sum_i t_i b_i \in R$.
- A vector $u(t) = (u(t)_1, \ldots, u(t)_n) \in R^n$ is defined such that each $u(t)_i$ is a linear form in the variables $t_1, \ldots, t_d$ and $\sum_i u(t)_i = \langle t, b \rangle$. Indeed, it is sufficient to define $u(t)_n = \langle t, b \rangle - \sum_{i=1}^{n-1} u(t)_i$, while the other entries of $u(t)$ can be chosen arbitrarily.
- Alice also computes a control vector $v(t) = u(t) + c$, where by definition $c = (c_1, \ldots, c_n) \in \mathbb{F}^n$ is a randomly chosen vector.
- Finally, she computes the values $\bar{u}(t) = F^{(t)}(u(t))$ and $\bar{v}(t) = F^{(t)}(v(t))$.
- The ciphertext computed by Alice for the plaintext $b = (b_1, \ldots, b_d)$ is defined as the triplet $\mathrm{CT}^{(t)}(b) = (\bar{u}(t), \bar{v}(t), c)$.

Functional encryption

- Alice computes the linear form $\langle t, b \rangle = \sum_i t_i b_i \in R$.
- A vector $u(t) = (u(t)_1, \ldots, u(t)_n) \in R^n$ is defined such that each $u(t)_i$ is a linear form in the variables $t_1, \ldots, t_d$ and $\sum_i u(t)_i = \langle t, b \rangle$. Indeed, it is sufficient to define $u(t)_n = \langle t, b \rangle - \sum_{i=1}^{n-1} u(t)_i$, while the other entries of $u(t)$ can be chosen arbitrarily.
- Alice also computes a control vector $v(t) = u(t) + c$, where by definition $c = (c_1, \ldots, c_n) \in \mathbb{F}^n$ is a randomly chosen vector.
- Finally, she computes the values $\bar{u}(t) = F^{(t)}(u(t))$ and $\bar{v}(t) = F^{(t)}(v(t))$.
- The ciphertext computed by Alice for the plaintext $b = (b_1, \ldots, b_d)$ is defined as the triplet $\mathrm{CT}^{(t)}(b) = (\bar{u}(t), \bar{v}(t), c)$.

Functional encryption

- Alice computes the linear form $\langle t, b \rangle = \sum_i t_i b_i \in R$.
- A vector $u(t) = (u(t)_1, \ldots, u(t)_n) \in R^n$ is defined such that each $u(t)_i$ is a linear form in the variables $t_1, \ldots, t_d$ and $\sum_i u(t)_i = \langle t, b \rangle$. Indeed, it is sufficient to define $u(t)_n = \langle t, b \rangle - \sum_{i=1}^{n-1} u(t)_i$, while the other entries of $u(t)$ can be chosen arbitrarily.
- Alice also computes a control vector $v(t) = u(t) + c$, where by definition $c = (c_1, \ldots, c_n) \in \mathbb{F}^n$ is a randomly chosen vector.
- Finally, she computes the values $\bar{u}(t) = F^{(t)}(u(t))$ and $\bar{v}(t) = F^{(t)}(v(t))$.
- The ciphertext computed by Alice for the plaintext $b = (b_1, \ldots, b_d)$ is defined as the triplet $\mathrm{CT}^{(t)}(b) = (\bar{u}(t), \bar{v}(t), c)$.

Functional encryption

- Alice computes the linear form $\langle t, b \rangle = \sum_i t_i b_i \in R$.
- A vector $u(t) = (u(t)_1, \ldots, u(t)_n) \in R^n$ is defined such that each $u(t)_i$ is a linear form in the variables $t_1, \ldots, t_d$ and $\sum_i u(t)_i = \langle t, b \rangle$. Indeed, it is sufficient to define $u(t)_n = \langle t, b \rangle - \sum_{i=1}^{n-1} u(t)_i$, while the other entries of $u(t)$ can be chosen arbitrarily.
- Alice also computes a control vector $v(t) = u(t) + c$, where by definition $c = (c_1, \ldots, c_n) \in \mathbb{F}^n$ is a randomly chosen vector.
- Finally, she computes the values $\bar{u}(t) = F^{(t)}(u(t))$ and $\bar{v}(t) = F^{(t)}(v(t))$.
- The ciphertext computed by Alice for the plaintext $b = (b_1, \ldots, b_d)$ is defined as the triplet $\mathrm{CT}^{(t)}(b) = (\bar{u}(t), \bar{v}(t), c)$.

Functional decryption

- Bob receives a ciphertext from Alice, namely a triplet $\mathrm{CT}^{(t)}(b) = (\bar{u}(t), \bar{v}(t), c)$.
- The goal of the functional decryption by Bob is to determine the vector $u(a) \in \mathbb{F}^n$ since it holds that $\sum_i u_i(a) = \langle a, b \rangle$. This vector is a specific preimage of $\bar{u}(a) \in \mathbb{F}^m$ that Bob aims to identify with the help of the difference $v(a) - u(a) = c$.
- By evaluating $t = (t_1, \ldots, t_d) \in R^d$ at the vector $a = (a_1, \ldots, a_d) \in \mathbb{F}^d$, Bob calculates $\bar{u} = \bar{u}(a)$ and $\bar{v} = \bar{v}(a)$.
- Using the user secret key $(F^{(a)}, O(a))$, Bob computes two preimages $u, v \in \mathbb{F}^n$ corresponding to $\bar{u}, \bar{v} \in \mathbb{F}^m$ with respect to the oil-vinegar map $F^{(a)} : \mathbb{F}^n \to \mathbb{F}^m$.
- Bob performs the test $v - u = c$ to verify whether $u = u(a)$. If this condition is satisfied, the functional decryption outputs $\sum_i u_i = \sum_i u_i(a) = \langle a, b \rangle$.
- If the test fails, another pair of preimages $u', v' \in \mathbb{F}^n$ are computed such that $F^{(a)}(u') = \bar{u}, F^{(a)}(v') = \bar{v}$.

Functional decryption

- Bob receives a ciphertext from Alice, namely a triplet $\mathrm{CT}^{(t)}(b) = (\bar{u}(t), \bar{v}(t), c)$.
- The goal of the functional decryption by Bob is to determine the vector $u(a) \in \mathbb{F}^n$ since it holds that $\sum_i u_i(a) = \langle a, b \rangle$. This vector is a specific preimage of $\bar{u}(a) \in \mathbb{F}^m$ that Bob aims to identify with the help of the difference $v(a) - u(a) = c$.
- By evaluating $t = (t_1, \ldots, t_d) \in R^d$ at the vector $a = (a_1, \ldots, a_d) \in \mathbb{F}^d$, Bob calculates $\bar{u} = \bar{u}(a)$ and $\bar{v} = \bar{v}(a)$.
- Using the user secret key $(F^{(a)}, O(a))$, Bob computes two preimages $u, v \in \mathbb{F}^n$ corresponding to $\bar{u}, \bar{v} \in \mathbb{F}^m$ with respect to the oil-vinegar map $F^{(a)} : \mathbb{F}^n \to \mathbb{F}^m$.
- Bob performs the test $v - u = c$ to verify whether $u = u(a)$. If this condition is satisfied, the functional decryption outputs $\sum_i u_i = \sum_i u_i(a) = \langle a, b \rangle$.
- If the test fails, another pair of preimages $u', v' \in \mathbb{F}^n$ are computed such that $F^{(a)}(u') = \bar{u}, F^{(a)}(v') = \bar{v}$.

Functional decryption

- Bob receives a ciphertext from Alice, namely a triplet $\mathrm{CT}^{(t)}(b) = (\bar{u}(t), \bar{v}(t), c)$.
- The goal of the functional decryption by Bob is to determine the vector $u(a) \in \mathbb{F}^n$ since it holds that $\sum_i u_i(a) = \langle a, b \rangle$. This vector is a specific preimage of $\bar{u}(a) \in \mathbb{F}^m$ that Bob aims to identify with the help of the difference $v(a) - u(a) = c$.
- By evaluating $t = (t_1, \ldots, t_d) \in R^d$ at the vector $a = (a_1, \ldots, a_d) \in \mathbb{F}^d$, Bob calculates $\bar{u} = \bar{u}(a)$ and $\bar{v} = \bar{v}(a)$.
- Using the user secret key $(F^{(a)}, O(a))$, Bob computes two preimages $u, v \in \mathbb{F}^n$ corresponding to $\bar{u}, \bar{v} \in \mathbb{F}^m$ with respect to the oil-vinegar map $F^{(a)} : \mathbb{F}^n \to \mathbb{F}^m$.
- Bob performs the test $v - u = c$ to verify whether $u = u(a)$. If this condition is satisfied, the functional decryption outputs $\sum_i u_i = \sum_i u_i(a) = \langle a, b \rangle$.
- If the test fails, another pair of preimages $u', v' \in \mathbb{F}^n$ are computed such that $F^{(a)}(u') = \bar{u}, F^{(a)}(v') = \bar{v}$.

Functional decryption

- Bob receives a ciphertext from Alice, namely a triplet $\mathrm{CT}^{(t)}(b) = (\bar{u}(t), \bar{v}(t), c)$.
- The goal of the functional decryption by Bob is to determine the vector $u(a) \in \mathbb{F}^n$ since it holds that $\sum_i u_i(a) = \langle a, b \rangle$. This vector is a specific preimage of $\bar{u}(a) \in \mathbb{F}^m$ that Bob aims to identify with the help of the difference $v(a) - u(a) = c$.
- By evaluating $t = (t_1, \ldots, t_d) \in R^d$ at the vector $a = (a_1, \ldots, a_d) \in \mathbb{F}^d$, Bob calculates $\bar{u} = \bar{u}(a)$ and $\bar{v} = \bar{v}(a)$.
- Using the user secret key $(F^{(a)}, O(a))$, Bob computes two preimages $u, v \in \mathbb{F}^n$ corresponding to $\bar{u}, \bar{v} \in \mathbb{F}^m$ with respect to the oil-vinegar map $F^{(a)} : \mathbb{F}^n \to \mathbb{F}^m$.
- Bob performs the test $v - u = c$ to verify whether $u = u(a)$. If this condition is satisfied, the functional decryption outputs $\sum_i u_i = \sum_i u_i(a) = \langle a, b \rangle$.
- If the test fails, another pair of preimages $u', v' \in \mathbb{F}^n$ are computed such that $F^{(a)}(u') = \bar{u}, F^{(a)}(v') = \bar{v}$.

Functional decryption

- Bob receives a ciphertext from Alice, namely a triplet $\mathrm{CT}^{(t)}(b) = (\bar{u}(t), \bar{v}(t), c)$.
- The goal of the functional decryption by Bob is to determine the vector $u(a) \in \mathbb{F}^n$ since it holds that $\sum_i u_i(a) = \langle a, b \rangle$. This vector is a specific preimage of $\bar{u}(a) \in \mathbb{F}^m$ that Bob aims to identify with the help of the difference $v(a) - u(a) = c$.
- By evaluating $t = (t_1, \ldots, t_d) \in R^d$ at the vector $a = (a_1, \ldots, a_d) \in \mathbb{F}^d$, Bob calculates $\bar{u} = \bar{u}(a)$ and $\bar{v} = \bar{v}(a)$.
- Using the user secret key $(F^{(a)}, O(a))$, Bob computes two preimages $u, v \in \mathbb{F}^n$ corresponding to $\bar{u}, \bar{v} \in \mathbb{F}^m$ with respect to the oil-vinegar map $F^{(a)} : \mathbb{F}^n \to \mathbb{F}^m$.
- Bob performs the test $v - u = c$ to verify whether $u = u(a)$. If this condition is satisfied, the functional decryption outputs $\sum_i u_i = \sum_i u_i(a) = \langle a, b \rangle$.
- If the test fails, another pair of preimages $u', v' \in \mathbb{F}^n$ are computed such that $F^{(a)}(u') = \bar{u}, F^{(a)}(v') = \bar{v}$.

Functional decryption

- Bob receives a ciphertext from Alice, namely a triplet $\mathrm{CT}^{(t)}(b) = (\bar{u}(t), \bar{v}(t), c)$.
- The goal of the functional decryption by Bob is to determine the vector $u(a) \in \mathbb{F}^n$ since it holds that $\sum_i u_i(a) = \langle a, b \rangle$. This vector is a specific preimage of $\bar{u}(a) \in \mathbb{F}^m$ that Bob aims to identify with the help of the difference $v(a) - u(a) = c$.
- By evaluating $t = (t_1, \ldots, t_d) \in R^d$ at the vector $a = (a_1, \ldots, a_d) \in \mathbb{F}^d$, Bob calculates $\bar{u} = \bar{u}(a)$ and $\bar{v} = \bar{v}(a)$.
- Using the user secret key $(F^{(a)}, O(a))$, Bob computes two preimages $u, v \in \mathbb{F}^n$ corresponding to $\bar{u}, \bar{v} \in \mathbb{F}^m$ with respect to the oil-vinegar map $F^{(a)} : \mathbb{F}^n \to \mathbb{F}^m$.
- Bob performs the test $v - u = c$ to verify whether $u = u(a)$. If this condition is satisfied, the functional decryption outputs $\sum_i u_i = \sum_i u_i(a) = \langle a, b \rangle$.
- If the test fails, another pair of preimages $u', v' \in \mathbb{F}^n$ are computed such that $F^{(a)}(u') = \bar{u}, F^{(a)}(v') = \bar{v}$.

Functional decryption

- Bob receives a ciphertext from Alice, namely a triplet $\mathrm{CT}^{(t)}(b) = (\bar{u}(t), \bar{v}(t), c)$.
- The goal of the functional decryption by Bob is to determine the vector $u(a) \in \mathbb{F}^n$ since it holds that $\sum_i u_i(a) = \langle a, b \rangle$. This vector is a specific preimage of $\bar{u}(a) \in \mathbb{F}^m$ that Bob aims to identify with the help of the difference $v(a) - u(a) = c$.
- By evaluating $t = (t_1, \ldots, t_d) \in R^d$ at the vector $a = (a_1, \ldots, a_d) \in \mathbb{F}^d$, Bob calculates $\bar{u} = \bar{u}(a)$ and $\bar{v} = \bar{v}(a)$.
- Using the user secret key $(F^{(a)}, O(a))$, Bob computes two preimages $u, v \in \mathbb{F}^n$ corresponding to $\bar{u}, \bar{v} \in \mathbb{F}^m$ with respect to the oil-vinegar map $F^{(a)} : \mathbb{F}^n \to \mathbb{F}^m$.
- Bob performs the test $v - u = c$ to verify whether $u = u(a)$. If this condition is satisfied, the functional decryption outputs $\sum_i u_i = \sum_i u_i(a) = \langle a, b \rangle$.
- If the test fails, another pair of preimages $u', v' \in \mathbb{F}^n$ are computed such that $F^{(a)}(u') = \bar{u}, F^{(a)}(v') = \bar{v}$.

Efficiency and security

- According to the UOV protocol, Bob's knowledge of the oil subspace $O(a)$ enables him to efficiently compute an element of any preimage of the oil-vinegar map $F(a)$.
- Indeed, we recall that such a computation reduces to solving a square system of linear equations.
- Assuming that the polynomial map $F^{(a)}$ is sufficiently generic, the dimension of a preimage, as an algebraic variety, is $n - m$.
- Consequently, the dimension of the product of preimages $(F^{(a)})^{-1}(\bar{u}) \times (F^{(a)})^{-1}(\bar{v})$ is $d = 2n - 2m$.
- Under the assumption that n is slightly larger than $2m$, we obtain that $d \approx n$.
- By imposing the n linear equations corresponding to the condition $v - u = c$ on the pairs $(u, v) \in (F^{(a)})^{-1}(\bar{u}) \times (F^{(a)})^{-1}(\bar{v})$, one essentially has a unique solution that coincides with the pair $(u(a), v(a))$. This guarantees correctness of decryption.

Efficiency and security

- According to the UOV protocol, Bob's knowledge of the oil subspace $O(a)$ enables him to efficiently compute an element of any preimage of the oil-vinegar map $F(a)$.
- Indeed, we recall that such a computation reduces to solving a square system of linear equations.
- Assuming that the polynomial map $F^{(a)}$ is sufficiently generic, the dimension of a preimage, as an algebraic variety, is $n - m$.
- Consequently, the dimension of the product of preimages $(F^{(a)})^{-1}(\bar{u}) \times (F^{(a)})^{-1}(\bar{v})$ is $d = 2n - 2m$.
- Under the assumption that n is slightly larger than $2m$, we obtain that $d \approx n$.
- By imposing the n linear equations corresponding to the condition $v - u = c$ on the pairs $(u, v) \in (F^{(a)})^{-1}(\bar{u}) \times (F^{(a)})^{-1}(\bar{v})$, one essentially has a unique solution that coincides with the pair $(u(a), v(a))$. This guarantees correctness of decryption.

Efficiency and security

- According to the UOV protocol, Bob's knowledge of the oil subspace $O(a)$ enables him to efficiently compute an element of any preimage of the oil-vinegar map $F(a)$.
- Indeed, we recall that such a computation reduces to solving a square system of linear equations.
- Assuming that the polynomial map $F^{(a)}$ is sufficiently generic, the dimension of a preimage, as an algebraic variety, is $n - m$.
- Consequently, the dimension of the product of preimages $(F^{(a)})^{-1}(\bar{u}) \times (F^{(a)})^{-1}(\bar{v})$ is $d = 2n - 2m$.
- Under the assumption that n is slightly larger than $2m$, we obtain that $d \approx n$.
- By imposing the n linear equations corresponding to the condition $v - u = c$ on the pairs $(u, v) \in (F^{(a)})^{-1}(\bar{u}) \times (F^{(a)})^{-1}(\bar{v})$, one essentially has a unique solution that coincides with the pair $(u(a), v(a))$. This guarantees correctness of decryption.

Efficiency and security

- According to the UOV protocol, Bob's knowledge of the oil subspace $O(a)$ enables him to efficiently compute an element of any preimage of the oil-vinegar map $F(a)$.
- Indeed, we recall that such a computation reduces to solving a square system of linear equations.
- Assuming that the polynomial map $F^{(a)}$ is sufficiently generic, the dimension of a preimage, as an algebraic variety, is $n - m$.
- Consequently, the dimension of the product of preimages $(F^{(a)})^{-1}(\bar{u}) \times (F^{(a)})^{-1}(\bar{v})$ is $d = 2n - 2m$.
- Under the assumption that n is slightly larger than $2m$, we obtain that $d \approx n$.
- By imposing the n linear equations corresponding to the condition $v - u = c$ on the pairs $(u, v) \in (F^{(a)})^{-1}(\bar{u}) \times (F^{(a)})^{-1}(\bar{v})$, one essentially has a unique solution that coincides with the pair $(u(a), v(a))$. This guarantees correctness of decryption.

Efficiency and security

- According to the UOV protocol, Bob's knowledge of the oil subspace $O(a)$ enables him to efficiently compute an element of any preimage of the oil-vinegar map $F(a)$.
- Indeed, we recall that such a computation reduces to solving a square system of linear equations.
- Assuming that the polynomial map $F^{(a)}$ is sufficiently generic, the dimension of a preimage, as an algebraic variety, is $n - m$.
- Consequently, the dimension of the product of preimages $(F^{(a)})^{-1}(\bar{u}) \times (F^{(a)})^{-1}(\bar{v})$ is $d = 2n - 2m$.
- Under the assumption that n is slightly larger than $2m$, we obtain that $d \approx n$.
- By imposing the n linear equations corresponding to the condition $v - u = c$ on the pairs $(u, v) \in (F^{(a)})^{-1}(\bar{u}) \times (F^{(a)})^{-1}(\bar{v})$, one essentially has a unique solution that coincides with the pair $(u(a), v(a))$. This guarantees correctness of decryption.

Efficiency and security

- According to the UOV protocol, Bob's knowledge of the oil subspace $O(a)$ enables him to efficiently compute an element of any preimage of the oil-vinegar map $F(a)$.
- Indeed, we recall that such a computation reduces to solving a square system of linear equations.
- Assuming that the polynomial map $F^{(a)}$ is sufficiently generic, the dimension of a preimage, as an algebraic variety, is $n - m$.
- Consequently, the dimension of the product of preimages $(F^{(a)})^{-1}(\bar{u}) \times (F^{(a)})^{-1}(\bar{v})$ is $d = 2n - 2m$.
- Under the assumption that n is slightly larger than $2m$, we obtain that $d \approx n$.
- By imposing the n linear equations corresponding to the condition $v - u = c$ on the pairs $(u, v) \in (F^{(a)})^{-1}(\bar{u}) \times (F^{(a)})^{-1}(\bar{v})$, one essentially has a unique solution that coincides with the pair $(u(a), v(a))$. This guarantees correctness of decryption.

Efficiency and security

- According to the UOV protocol, Bob's knowledge of the oil subspace $O(a)$ enables him to efficiently compute an element of any preimage of the oil-vinegar map $F(a)$.
- Indeed, we recall that such a computation reduces to solving a square system of linear equations.
- Assuming that the polynomial map $F^{(a)}$ is sufficiently generic, the dimension of a preimage, as an algebraic variety, is $n - m$.
- Consequently, the dimension of the product of preimages $(F^{(a)})^{-1}(\bar{u}) \times (F^{(a)})^{-1}(\bar{v})$ is $d = 2n - 2m$.
- Under the assumption that n is slightly larger than $2m$, we obtain that $d \approx n$.
- By imposing the n linear equations corresponding to the condition $v - u = c$ on the pairs $(u, v) \in (F^{(a)})^{-1}(\bar{u}) \times (F^{(a)})^{-1}(\bar{v})$, one essentially has a unique solution that coincides with the pair $(u(a), v(a))$. This guarantees correctness of decryption.

- According to the simulation-based paradigm, the confidentiality of the IPFE protocol is ensured by the security of the UOV protocol.
- An experimental verification of this IPFE protocol is currently in progress, focusing on case studies of practical significance.

- According to the simulation-based paradigm, the confidentiality of the IPFE protocol is ensured by the security of the UOV protocol.
- An experimental verification of this IPFE protocol is currently in progress, focusing on case studies of practical significance.

Bibliography

- Beullens, W.; Chen, M.-S.; Ding, J.; Gong, B.; Kannwischer, M.J.; Patarin, J.; Peng, B.-Y.; Schmidt, D.; Shih, C.-J.; Tao, C.; Yang, B.-Y., UOV: Unbalanced Oil and Vinegar Algorithm Specifications and Supporting Documentation, Version 1.0, 2023.
- Boneh, D.; Sahai, A.; Waters, B., Functional encryption: definitions and challenges. Theory of cryptography, 253 – 273. Lecture Notes in Comput. Sci., 6597, Springer, Heidelberg, 2011.
- Debnath, S.K.; Mesnager, S.; Dey, K.; Kundu, N., Post-quantum secure inner product functional encryption using multivariate public key cryptography. Mediterr. J. Math. 18 (2021), no. 5, Paper No. 204, 15 pp.
- La Scala, R., A protocol for inner product functional encryption based on the UOV scheme, preprint, 2024.
- Mascia, C.; Sala, M.; Villa, I., A survey on functional encryption. Adv. Math. Commun. 17 (2023), no. 5, 1251–1289.

Bibliography

- Beullens, W.; Chen, M.-S.; Ding, J.; Gong, B.; Kannwischer, M.J.; Patarin, J.; Peng, B.-Y.; Schmidt, D.; Shih, C.-J.; Tao, C.; Yang, B.-Y., UOV: Unbalanced Oil and Vinegar Algorithm Specifications and Supporting Documentation, Version 1.0, 2023.
- Boneh, D.; Sahai, A.; Waters, B., Functional encryption: definitions and challenges. Theory of cryptography, 253 – 273. Lecture Notes in Comput. Sci., 6597, Springer, Heidelberg, 2011.
- Debnath, S.K.; Mesnager, S.; Dey, K.; Kundu, N., Post-quantum secure inner product functional encryption using multivariate public key cryptography. Mediterr. J. Math. 18 (2021), no. 5, Paper No. 204, 15 pp.
- La Scala, R., A protocol for inner product functional encryption based on the UOV scheme, preprint, 2024.
- Mascia, C.; Sala, M.; Villa, I., A survey on functional encryption. Adv. Math. Commun. 17 (2023), no. 5, 1251–1289.