

# Sharing Privacy: From 0 to PSI (Private Set Intersection)

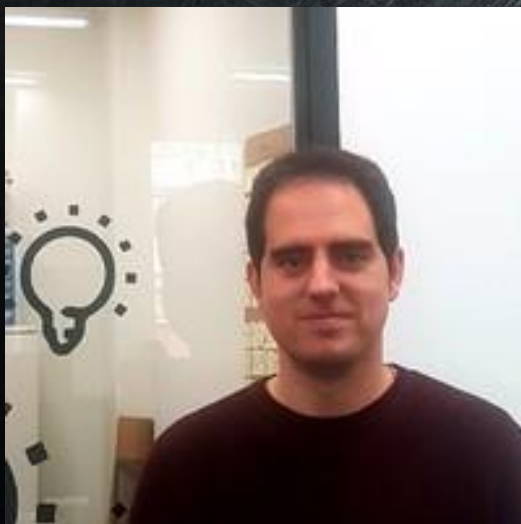
Dr. Alfonso Muñoz (@mindcrypt)

José Ignacio Escribano

RootedCon Valencia 2019

# \$whoami

Dr. Alfonso Muñoz  
alfonso@criptored.com  
alfonso.munoz2.next@bbva.com  
@mindcrypt  
Telegram: t.me/Criptored  
Linkedin: <https://www.linkedin.com/in/alfonsomuñoz/>



- Global Technical Cybersecurity Lead & Head of Cybersecurity – BBVA Next Technologies
- **Vidas paralelas**
  - Red Temática Criptored
  - Expert security member Europol
  - Criptocert – <https://www.criptocert.com>
  - Libros, certificaciones, conferencias, tools, instructor...
- **Áreas**
  - Offensive security (sw/hw)
  - Cryptography & Covert channels/Stego
  - Cutting-edge research (defensive & Offensive)



# \$whoami

José Ignacio Escribano

joseignacio.escribano.pablos.next@bbva.com



- Security & Machine Learning Researcher – BBVA Next Technologies
- **Vidas paralelas**
  - Graduado en Matemáticas e Ingeniería del Software & Máster en Ingeniería de la Decisión.
  - Actualmente, realizando tesis sobre criptografía post-cuántica.
  - Ponente en Cybercamp y Hack In The Box
- **Áreas**
  - Inteligencia Artificial
  - Criptografía
  - Cutting-edge research (Offensive & defensive)

# Agenda para la charla



Objetivo de la charla. Introducción



Criptografía (full) homomórfica

- Definiciones. Librerías y seguridad REAL
- Demo con librería SEAL



Secure Multiparty Computation (SMC)

- Private Set Intersection (2PC & MPC)
  - Demo: Intercambio anónimo de agendas de contacto app de mensajería (privacidad)
- Private Intersection-Sum (Google)



Conclusiones



# El mejor ataque... una buena defensa

- Seguridad Perimetral



VS



- Insiders

# El atacante está dentro... ¿Ahora qué?

- Tecnología Zero-Trust
  - *"Los CIO deben avanzar hacia un enfoque Zero Trust para la seguridad centrada en los datos y la identidad, siendo desde nuestro punto de vista el único enfoque de seguridad que funciona..."*
- Deception technology – “Alimentemos a la bicha”
- Criptografía – Protección de datos y ¿gestión del ciclo de vida de las claves?

“If you think technology can solve your security problems, then you don't understand the problems and you don't understand the technology.”

Bruce Schneier



# Defectos de la criptografía (I/II)

- **Criptografía mal implementada** (precauciones)
  - Crear tu propio algoritmo o implementar uno existente
  - Mal uso de librerías o algoritmos
    - <https://security.googleblog.com/2018/08/introducing-tink-cryptographic-software.html>
    - <https://github.com/google/wycheproof>
- Incorrecta protección de claves criptográficas
- Aleatoriedad que no es aleatoria
- Criptografía no aislada
  - Ataques de canal lateral. Programación en tiempo constante
  - Ataques de compresión (BREACH, TIME, HEIST, FIESTA...)
  - Ataques basados en downgrade y flujos del protocolo (Logjam, Freak, 3SHAKE)
  - Ataques basados en relleno/padding (Lucky13, Drown, Robot, New Bleichenbacher side channels, Poodle, Zombie Poodle, Golden Poodle)





# Defectos de la criptografía (II/II)

- Ciclo de vida de la gestión de claves: las claves se roban típicamente malware o ataques de canal lateral (Meltdown, Spectre, ...)
  - <https://i.blackhat.com/us-18/Wed-August-8/us-18-Guri-AirGap.pdf>
- La información/clave en algún momento está en claro...
- ¿ALTERNATIVA? – Trabajar en el **dominio cifrado**
  - Criptografía homomórfica (HE)
  - Computación multiparte segura (MPC)
  - Searchable Encryption (SE) / Whitebox Cryptography
  - ~~IA en criptografía~~



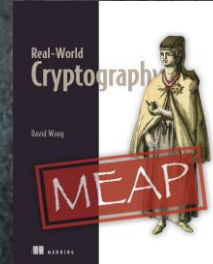
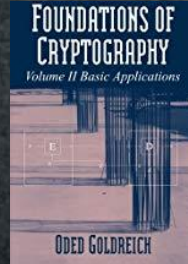
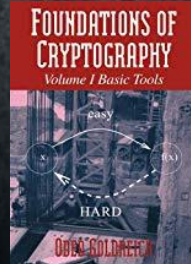
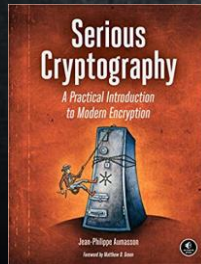
Criptografía, Deep Learning y Google. Desmontando un sueño. Alfonso Muñoz – Navaja Negra 2018 - <https://www.youtube.com/watch?v=VizSx4dwJMw>



# Homework - : )

- Receta para el buen (crypto) analista/arquitecto/investigador...
  - Recomendación algoritmos/librerías: Cryptographic Best Practices (Aaron Toponce)
   
<https://gist.github.com/atoponce/07d8d4c833873be2f68c34f9afc5a78a>
  
<https://safecurves.cr.yp.to/>
  - Asesórate bien (¿stackoverflow?)
  - Cursos:
    - <https://es.coursera.org/learn/crypto>
    - <http://www.criptored.upm.es/crypt4you/portada.html>
  - Certificación: (descuento 15% - CRÎPTOCERTROOTEDCONVLC2019)
    - <https://www.cryptocert.com>

- Libros:



**Stack Overflow Considered Helpful!**  
**Deep Learning Security Nudges Towards Stronger Cryptography**  
<https://www.usenix.org/system/files/sec19-fischer.pdf>

Felix Fischer, Huang Xiao<sup>†</sup>, Ching-Yu Kao\*, Yannick Stachelscheid, Benjamin Johnson, Danial Razar, Paul Fawkesley<sup>‡</sup>, Nat Buckley<sup>‡</sup>, Konstantin Böttinger<sup>‡</sup>, Paul Muntean, Jens Grossklags  
*Technical University of Munich, <sup>†</sup>Bosch Center for Artificial Intelligence*  
*\*Fraunhofer AISEC, <sup>‡</sup>Projects by IF*

{f.fischer, yannick.stachelscheid, benjamin.johnson, danial.razar, paul.muntean, jens.grossklags}@tum.de  
 {huang.xiao}@de.bosch.com, {nat, paul}@projectsbyif.com, {ching-yu.kao, konstantin.boettinger}@aisec.fraunhofer.de

**Abstract**

Stack Overflow is the most popular discussion platform for software developers. However, recent research identified a large amount of insecure encryption code in production systems that has been inspired by examples given on Stack Overflow. By copying and pasting functional code, developers introduced exploitable software vulnerabilities into security-sensitive high-profile applications installed by millions of users every day. Proposed mitigations of this problem suffer from usability flaws and push developers to continue shopping for code examples on Stack Overflow once again. This motivates us to fight the proliferation of insecure code directly at the root before it even reaches the clipboard. By viewing Stack Overflow as a market, implementation of cryptography becomes a decision-making problem. In this context, our goal is to simplify the selection of helpful and secure examples. More specifically, we focus on supporting software developers in making better decisions on Stack Overflow by applying nudges, a concept borrowed from behavioral economics and psychology. This approach is motivated by one of our key findings: For 99.37% of insecure code examples on Stack Overflow, similar alternatives are available that serve the same use case and provide strong cryptography. Our system design that modifies Stack Overflow is based on several nudges that are controlled by a deep neural network. It learns a representation for cryptographic API usage patterns and classification of their security, achieving average AUC-ROC of 0.992. With a user study, we demonstrate that nudge-based security advice significantly helps tackling the most popular and error-prone cryptographic use cases in Android.

The fact that 78% of software developers primarily seek help on Stack Overflow on a daily basis<sup>1</sup> underlines the usability and perceived value of community and example-driven documentation [2].

Reuse of code examples is the most frequently observed user pattern on Stack Overflow [17]. It reduces the effort for implementing a functional solution to its minimum and the functionality of the solution can immediately be tested and verified. However, when implementing encryption, its security, being a non-functional property, is difficult to verify as it necessitates profound knowledge of the underlying cryptographic concepts. Moreover, most developers are unaware of pitfalls when applying cryptography and that misuse can actually harm application security. Instead, it is often assumed that mere application of any encryption is already enough to protect private data [13, 14]. Stack Overflow users also cannot rely on the community to correctly verify the security of available code examples [9]. Security advice given by community members and moderators is mostly missing and oftentimes overlooked. This is due to only a few security experts being available as community moderators and a feedback system which is not sufficient to communicate security advice effectively. Consequently, highly insecure code examples are frequently reused in production code [17]. Exploiting these insecure samples, high-profile applications were successfully attacked, leading to theft of user credentials, credit card numbers and other private data [13].

While mainly focused on the negative impact of Stack Overflow on code security, recent research has also reported that there is a full range of code snippets providing strong security for symmetric, asymmetric and password-based encryption, as well as TLS, message digests, random number generation, and authentication [17]. However, it was previ-



# Dominio cifrado (for dummies) 😊

- Trabajar sobre “**textos**” **cifrados** en vez de “texto” plano.
- Distintas alternativas:
  - **Criptografía homomórfica (HE)**: permite hacer cálculos sobre textos cifrados (sin necesidad de descifrar).
  - **Searchable encryption (SE)**: típicamente permite realizar búsquedas sobre bases de datos cifradas.
  - **Computación multiparte segura (MPC)**: permite calcular una función de forma conjunta entre todas las partes sin relevar los datos de cada parte.
    - Distintas funciones: Problema de los millonarios (socialistas), Private Set Intersection (PSI), Private Intersection and Sum (PIS), ...

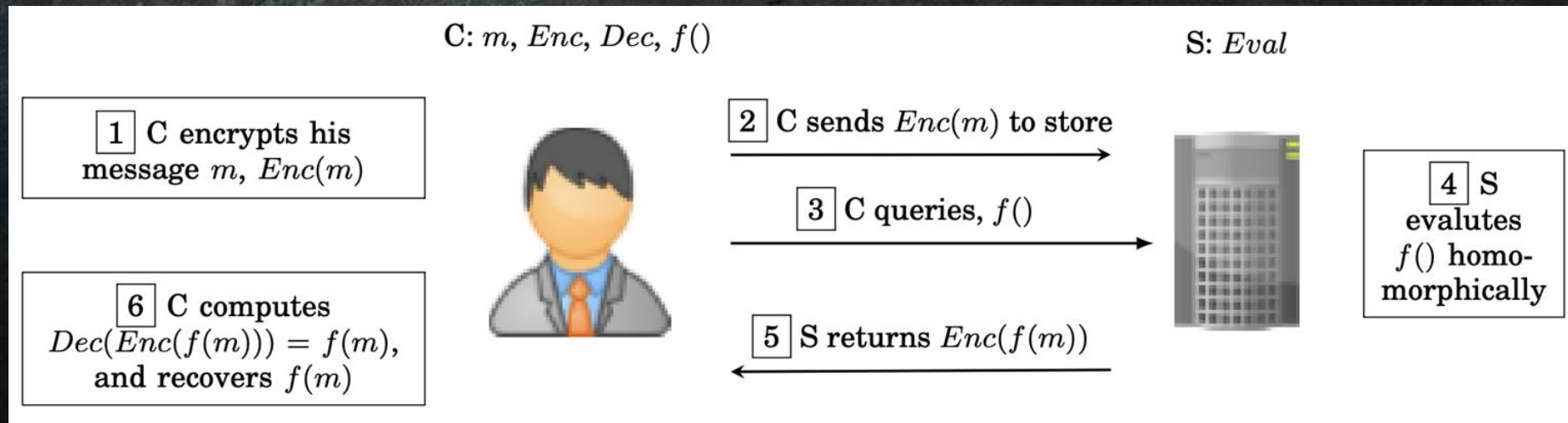


# Cifrado homomórfico



# Cifrado homomórfico

- Permite hacer **cálculos sobre textos cifrados** (sin necesidad de descifrar).



- Aplicación: **delegar cálculos en la nube.**



# Cifrado homomórfico (definición)

- Un cifrado es homomórfico (respecto a la operación  $*$ ) si

$$E(m1) * E(m2) = E(m1 * m2)$$

para cualquier par de mensajes  $m1$  y  $m2$ .  $E$  es la función de cifrado.

# Cifrado homomórfico (RSA)

- RSA es un cifrado homomórfico (parcial).

$$E(m) = m^e \pmod{n}$$


- Se cumple que para cualquier mensaje  $m_1, m_2$ ,


$$\begin{aligned} E(m_1) * E(m_2) &= (m_1^e \pmod{n}) * (m_2^e \pmod{n}) \\ &= (m_1 * m_2)^e \pmod{n} \\ &= E(m_1 * m_2). \end{aligned}$$



# Cifrado homomórfico (RSA)

$p = 5; q = 11; n = p * q = 55$   
 $\Phi(n) = (p-1) * (q-1) = 40$   
 $e = 7; d = 23$

  $(n=55, e=7)$

  $(n=55, d=23)$

$E(m) = m^e \bmod n = c$   
 $D(c) = c^d \bmod n$

$m1, m2$

calcular

$m1 * m2$

8, 3

cifrar

$E(8), E(3)$   
 2, 42

$E(m1), E(m2)$

calcular

$E(m1) * E(m2) = E(m1 * m2)$

$8 * 3$

cifrar

$E(8) * E(3) = E(8 * 3)$   
 $E(24) = 2 * 42 = 29 \bmod 55$   
 $E(8) * E(3) = E(8 * 3) = 29$



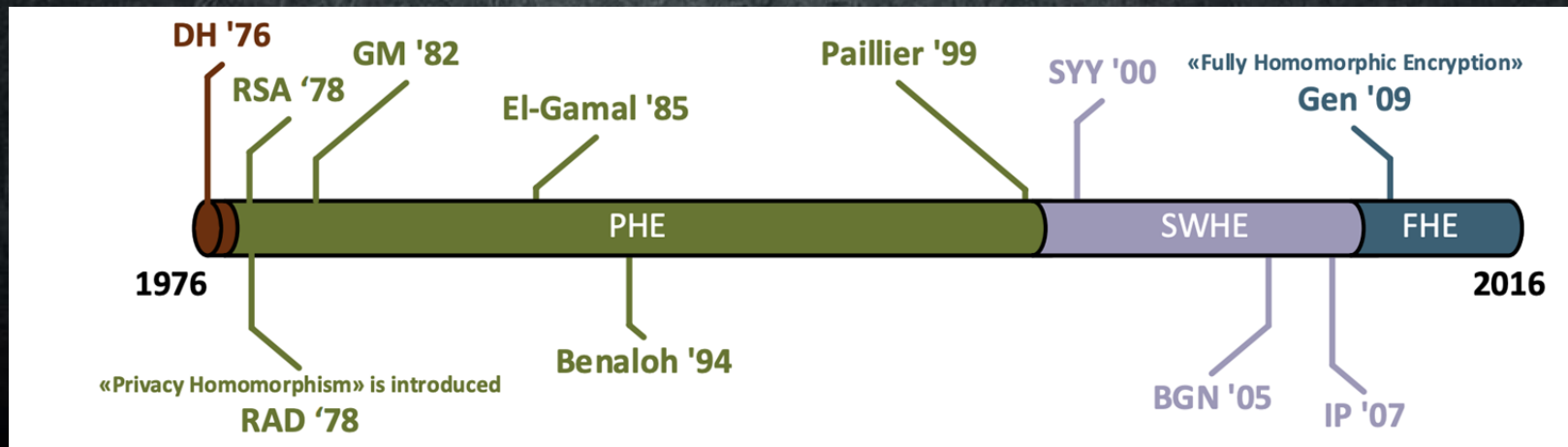
# Cifrado homomórfico (aplicaciones)

- **Salud:** mantener la privacidad de los pacientes es crítico, pero sin embargo, compartir y hacer cálculos sobre esa información es importante para distintos casos de uso como cuidado del paciente coordinado entre distintas entidades, fraude en las facturas, reembolso, etc.
- **Genómica:** los datos privados de la secuenciación del genoma humano pueden ser una herramienta muy potente en desarrollar una cura/tratamiento para una enfermedad. Existen problemas al compartir estos datos, ya que las secuencias del ADN individuales son únicas y permiten decir si un individuo es susceptible de tener una cierta enfermedad.
- **Servicios financieros:** en este ámbito se trabaja con información confidencial, por lo que datos, modelos y funciones calculadas sobre ellos son considerados confidenciales y propietarios. La criptografía homomórfica puede ayudar a ejecutar estas funciones de forma privada en la nube, reduciendo costes y tiempo.
- **Reto: casos de uso ...**



# Cifrado homomórfico (tipos)

- **Partially Homomorphic Encryption (PHE)**: sólo permite una operación homomórfica (típicamente suma o producto) un número ilimitado de veces.
  - Ejemplos: RSA, Paillier, El-Gamal, Goldwasser & Micali, “el cifrado César” ...
- **Somewhat Homomorphic Encryption (SWHE)**: permite realizar algunas operaciones un número limitado de veces.
- **Fully Homomorphic Encryption (FHE)**: permite realizar un número ilimitado de operaciones ilimitadas veces (“truco”).



# Cifrado homomórfico - Limitaciones

- Explicación “muy simple” (FHE): se añade ruido para al operar no poder conocer la información en claro ni las claves involucradas. **El ruido hay que eliminarlo**
  - Craig Gentry, *Fully homomorphic encryption using ideal lattices*, Symposium on the Theory of Computing (STOC), 2009, pp. 169-178. - <https://crypto.stanford.edu/craig/craig-thesis.pdf>
- **Eliminar ruido: Bootstrapping** - <https://crypto.stackexchange.com/Questions/42666/what-exactly-is-bootstrapping-in-fhe>
- Control del ruido (leveled homomorphic encryption, ring-LWE, etc.)
  - Esquemas acelerados (no se elimina el error mediante bootstrapping) pero número de operaciones acotado a un punto donde el error acumulado hace irre recuperable el mensaje
    - <https://crypto.stackexchange.com/questions/15794/difference-between-leveled-fhe-and-normal-fhe-scheme>
- Nuevas propuestas... (desparece la clave de evaluación, “desaparece el ruido”, ...) → jose verificar



# Cifrado homomórfico (tipos)

- **Distintos modelos de computación** homomórfica:
  - Circuitos booleanos.
  - Aritmética modular (exacta).
  - Aritmética de números (aproximada).
- Típicamente, cifrados homomórficos completos se pueden construir con el uso de **retículos** (lattices).
- La seguridad de los retículos viene dada por la dificultad de resolver los problemas (Ring) Learning With Errors **(R)LWE**.

EXPLICAR

LWE

4	1	11	10	x	6	+	0	=	4
5	5	9	5		9		-1		7
3	9	0	10		11		1		2
1	3	3	2		11		1		11
12	7	3	4		11		1		5
6	5	11	4		0		12		
3	3	5	0		-1		8		

RLWE

4	1	11	10	x	6	9	11	11	+	0	-1	1	1	=	10	5	10	7


# Cifrado homomórfico (librerías)

Nombre	Url	Tipo
HElib	<a href="https://github.com/HomEnc/HElib">https://github.com/HomEnc/HElib</a>	FHE
SEAL	<a href="http://sealcrypto.org/">http://sealcrypto.org/</a>	FHE
HEAAN	<a href="https://github.com/kimandrik/HEAAN">https://github.com/kimandrik/HEAAN</a>	FHE
TFHE	<a href="https://github.com/tfhe/tfhe">https://github.com/tfhe/tfhe</a>	FHE
libshe	<a href="https://github.com/bogdan-kulynych/libshe">https://github.com/bogdan-kulynych/libshe</a>	SWHE
cuHE	<a href="https://github.com/vernamlab/cuHE">https://github.com/vernamlab/cuHE</a>	FHE
python-paillier	<a href="https://github.com/n1analytics/python-paillier">https://github.com/n1analytics/python-paillier</a>	PHE
pycryptodome	<a href="https://pycryptodome.readthedocs.io">https://pycryptodome.readthedocs.io</a>	PHE

Ver <https://homomorphicencryption.org/introduction/> o <https://github.com/jonaschn/awesome-he#libraries> para más librerías.



# Cifrado homomórfico (librerías) - Desmitificando

Funcionalidad	SEAL	HElib	TFHE	Paillier	ElGamal	RSA
Cifrado asimétrico	✓	✓	✓	✓	✓	✓
Soporte para números negativos	✓	✗	✗	✗	✗	✗
Tamaño textos cifrados <1MB (1 entrada)	✗	✗	✓	✓	✓	✓
Puede ejecutarse en menos de 2GB RAM	✗	✓	✓	✓	✓	✓
Lenguajes disponibles	 	 		  		

# Cifrado homomórfico (librerías)

Operación	SEAL	HElib	TFHE	Paillier	ElGamal	RSA
Suma, resta	✓	✓	✓	✓	✓	✗
Multiplicación	✓	✓	✓	✗	✓	✓
Comparación, división	✗	✗	✗	✗	✗	✗
Operaciones booleanas	✗	✗	✓	✗	✗	✗
Ops. bit a bit, matriciales	✓	✓	✗	✗	✗	✗
Exponenciación	✓	✓	✗	✗	✗	✗
Elevar al cuadrado	✓	✓	✗	✗	✓	✗
Ops. texto cifrado con texto plano	✓	✗	✗	✗	✗	✗



# Cifrado homomórfico (SEAL)

- SEAL (<https://github.com/microsoft/SEAL>) es una librería de criptografía homomórfica desarrollada por **Microsoft**. Escrita en **C++**.
- Implementa los esquemas **BFV** y **CKKS**. Ambos esquemas **IND-CPA** (ver [Peng2019]).
- Cálculo de los **parámetros de forma automática** en función del nivel de seguridad fijado.
- **API sencilla** y fácil de entender.
- **Amplia documentación** y ejemplos disponibles:
  - <https://github.com/microsoft/SEAL/tree/master/native/examples>
  - <https://github.com/microsoft/SEAL-Demo>

# Cifrado homomórfico (demo)

- Aplicación de **subida de salario** de los empleados de una empresa.
- Desarrollada con **SEAL** (versión 3.2.0) en C++, usando el esquema **CKKS** [Cheon2017].
- Computa, de forma homomórfica, la función

$$\begin{aligned}s' &= (1 + 1/100 * p) * s \\ &= (1 + 0.01 * p) * s\end{aligned}$$

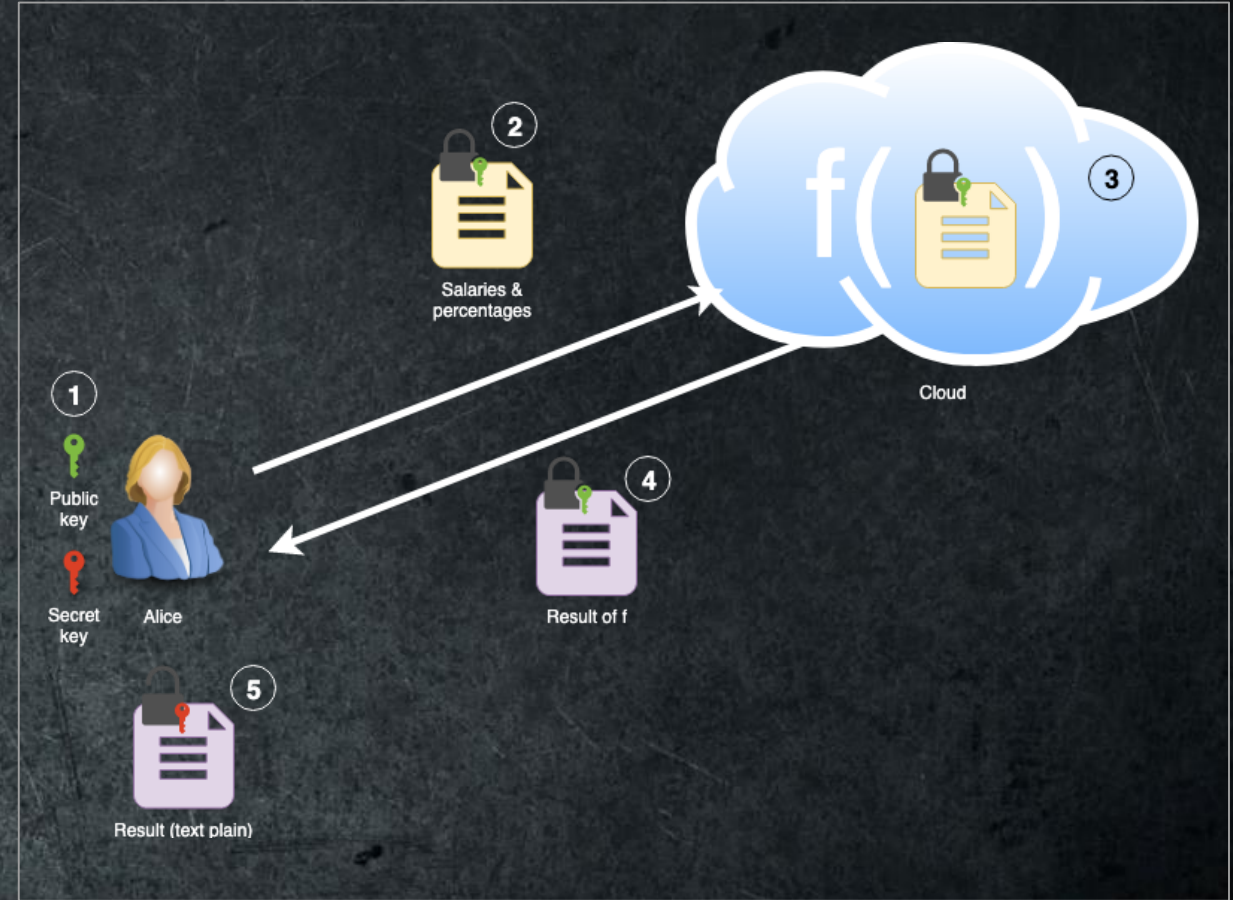
donde  $s'$  es el nuevo salario,  $s$  es el salario actual y  $p$  es el porcentaje de subida (en tanto por cien).

- Necesarias sólo **sumas y multiplicaciones**.

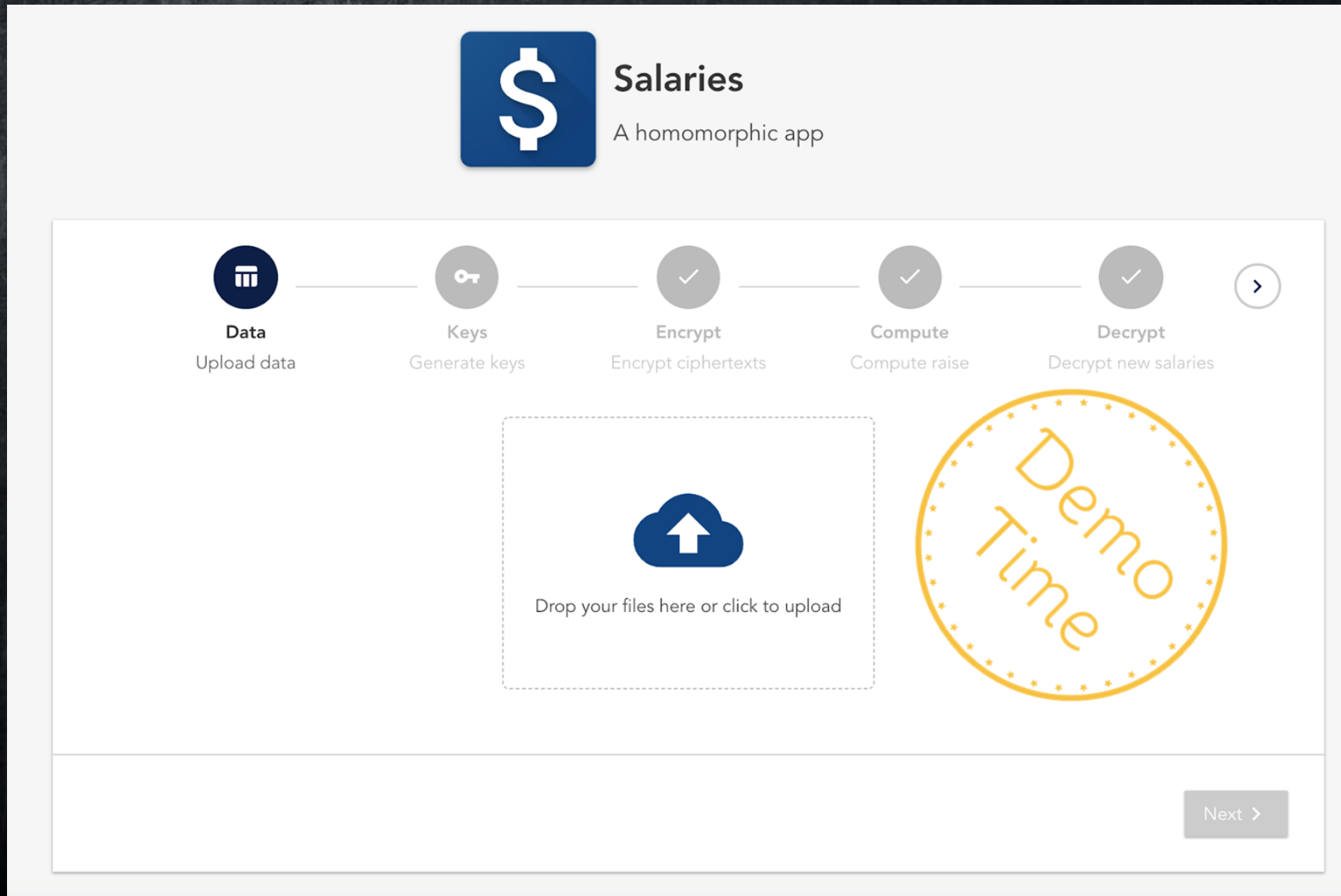


# Cifrado homomórfico (demo)

1. Alice genera un **par de claves** (pública y privada).
2. Alice **cifra los salarios y los porcentajes** de cada empleado con su clave pública y los envía a la nube.
3. La nube **computa la función de forma homomórfica**  $f(p,s) = (1+0.01*p)*s$ , que devuelve resultado cifrado de  $f$ .
4. La nube devuelve el resultado a Alice.
5. Alice **descifra el texto cifrado con su clave privada** y obtiene el resultado en claro de evaluar la función  $f$ .



# Cifrado homomórfico (demo)





# Cifrado homomórfico (demo)

```
Evaluator evaluator(context);  
CKKSEncoder encoder(context);
```

```
Plaintext plain_001, plain_1;  
encoder.encode(0.01, scale, plain_001);
```

```
// Computes  $(0.01 * \text{percentage})$   
evaluator.multiply_plain_inplace(encrypted_percentage, plain_001);  
evaluator.relinearize_inplace(encrypted_percentage, relin_keys);
```

```
// Computes  $(1 + 0.01 * \text{percentage})$   
encoder.encode(1, encrypted_percentage.scale(), plain_1);  
evaluator.add_plain_inplace(encrypted_percentage, plain_1);  
evaluator.relinearize_inplace(encrypted_percentage, relin_keys);
```

```
// Computes  $(1 + 0.01 * \text{percentage}) * \text{salary}$   
evaluator.rescale_to_inplace(encrypted_percentage, encrypted_salary.parms_id());  
evaluator.multiply_inplace(encrypted_salary, encrypted_percentage);
```



# Cifrado homomórfico (ataques)

- El **estándar de criptografía homomórfica** [Albrecht2018] recoge los ataques más comunes basados en LWE/RLWE.
  - uSVP
  - Bounded Distance Decoding (BDD)
  - Dual attack
  - BKZ
- Además, proporciona **tablas de seguridad** para fijar los parámetros en función del nivel de seguridad requerida.
- El código para reproducir las tablas de seguridad se puede encontrar en <https://bitbucket.org/malb/lwe-estimator/src/master/> (Ver [Albrecht2015]).

Danger of using fully homomorphic encryption: A look at Microsoft SEAL <https://arxiv.org/pdf/1906.07127.pdf>



# Tablas de seguridad

distribution	n	security level	log(q)	uSVP	dec	dual
uniform	1024	128	31	130.6	133.8	147.5
		192	22	203.6	211.2	231.8
		256	18	269.9	280.5	303.6
	2048	128	59	129.5	129.7	139.2
		192	42	194.0	197.6	212.4
		256	33	263.8	270.7	289.9
	4096	128	113	131.9	129.4	136.8
		192	80	192.7	193.2	203.2
		256	63	260.7	263.6	277.6
	8192	128	222	132.9	128.9	134.9
		192	157	195.4	192.8	200.6
		256	124	257.0	256.8	266.7
16384	16384	128	440	133.9	129.0	133.0
		192	310	196.4	192.4	198.7
		256	243	259.5	256.6	264.1
32768	32768	128	880	134.3	129.1	131.6
		192	612	198.8	193.9	198.2
		256	480	261.6	257.6	263.6

distribution	n	security level	log(q)	uSVP	dec	dual
(-1,1)	1024	128	29	139.6	145.9	128.9
		192	20	226.9	241.2	196.8
		256	15	344.3	366.1	273.9
	2048	128	56	136.2	137.9	128.3
		192	39	210.3	217.5	194.6
		256	30	294.5	307.5	268.8
	4096	128	110	135.1	133.5	128.5
		192	77	203.1	205.5	192.4
		256	60	274.7	280.4	259.0
	8192	128	219	134.6	130.9	128.6
		192	153	200.3	199.0	193.1
		256	119	268.7	270.0	257.9
	16384	128	441	133.3	128.7	128.1
		192	306	199.0	195.3	192.4

# ¿Qué librería usar?

- La elección de la mejor librería dependerá de:
  - El **caso de uso** a implementar.
  - **Nivel de seguridad** requerido.
  - **Tipo de seguridad** (IND-CPA, IND-CCA).
  - **Número de operaciones** requeridas.
  - **Velocidad**.
  - **Escalabilidad**.
  - Tipo de **modelo de computación** a emplear.



# MultiParty Computation + Private Set Intersection



# Secure Multiparty Computation (MPC)

- Permite **calcular una función de forma conjunta** entre todas las partes sin revelar los datos de cada parte.
- Diferentes funciones:
  - **Problema de los millonarios**: decidir qué millonario tiene mayor cantidad de dinero sin revelar sus fortunas. Es decir, calcular si  $a \geq b$ .
  - **Problema de los millonarios socialistas**: decidir si los millonarios tienen la misma cantidad de dinero, es decir, si  $a = b$ .
  - **Operaciones sobre conjuntos** [Kissner2005].
    - Subconjunto:  $A \subseteq B$ .
    - Cardinalidad:  $|A| = |B|$ .
    - Intersección (**PSI**):  $A \cap B$ .
  - Otros:
    - Private Intersection-Sum [Ion2019] (**PIS**): PSI + PHE (suma).
- Protocolos: ver [Evans2018].



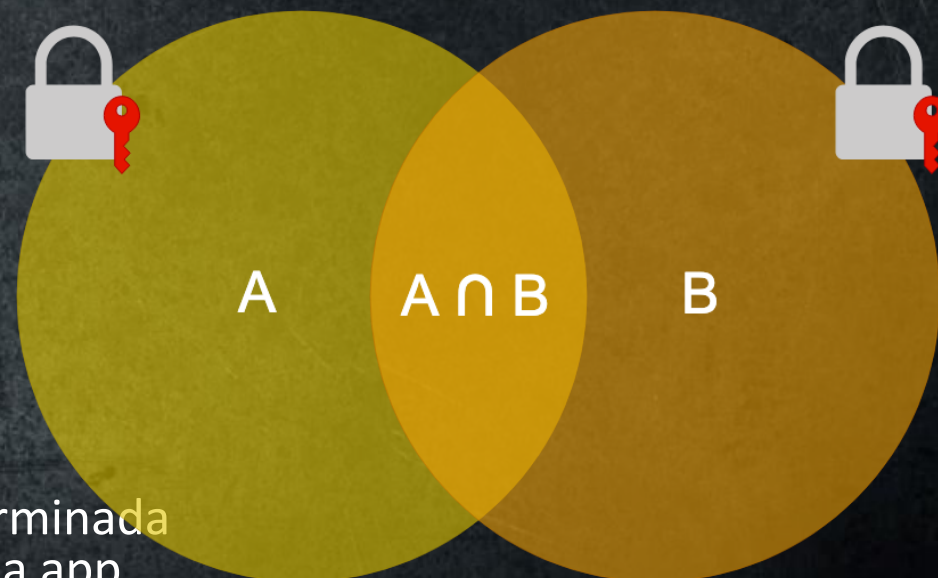
# Secure Multiparty Computation (MPC)

- Adversarios:
  - **Honesto pero curioso, semihonesto o pasivo**: el adversario sigue el protocolo especificado, pero trata de aprender lo máximo posible con los mensajes recibidos.
  - **Malicioso o activo**: el adversario puede desviarse arbitrariamente de la ejecución del protocolo para obtener una ventaja sobre alguna de las otras partes participantes del protocolo.
- Durante la presentación sólo consideraremos adversarios **semihonestos**.



# Private Set Intersection (PSI)

- Calcular los **elementos comunes** entre distintos conjuntos de datos sin revelar ninguna otra información entre las partes.
- Existen protocolos para sólo **2 partes** o **multiparte**.
- Casos de uso:
  - Una empresa gubernamental quiere estar segura de que los empleados que subcontrata no tienen antecedentes penales y ninguna de las empresas quiere ceder los datos: lista de ciudadanos con antecedentes ni lista de empleados, pero quieren conocer la intersección.
  - Hacienda quiere conocer a evasores de impuestos con cuentas en algún banco extranjero. El banco no quiere compartir su lista de clientes y Hacienda no revela su lista de sospechosos.
  - Un usuario tiene una lista de contactos en su móvil y quiere conocer qué contactos están usando una determinada aplicación de chat sin enviarle su lista de contactos a la app.





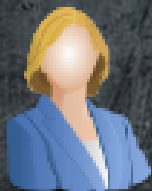
# Protocolo PSI (2 partes)

- Basado en que la hipótesis **DDH** (Decisional Diffie-Hellman) es **difícil computacionalmente**. - [https://en.wikipedia.org/wiki/Decisional\\_Diffie%E2%80%93Hellman\\_assumption](https://en.wikipedia.org/wiki/Decisional_Diffie%E2%80%93Hellman_assumption)
- Funciona aunque la **cardinalidad de los conjuntos sean distintas**.
- Seguro contra adversarios **semihonestos**.
- Requiere del **cifrado de exponenciación de Pohlig-Hellman** [Holden2008, Pohlig1978]. Es un cifrado **conmutativo**.
  - **Generación de clave**:  $k \leftarrow \mathbb{Z}_p$  (clave cifrado);  $d = k^{-1} \bmod p-1$  (clave descifrado)
  - **Cifrado**:  $E(m) = m^k \bmod p = c$
  - **Descifrado**:  $D(c) = c^d \bmod p$

[https://en.wikipedia.org/wiki/Pohlig%E2%80%93Hellman\\_algorithm](https://en.wikipedia.org/wiki/Pohlig%E2%80%93Hellman_algorithm)

# Protocolo PSI (2 partes)

Alice



V =

+34633946745
+34759421743
+34623434477
+34602305532

Bob



W =

+34687399815
+34671374657
+34759421743



# Protocolo PSI (2 partes)

Alice



V =

+34633946745
+34759421743
+34623434477
+34602305532

Información común: función hash  $H$  y  $p$

Bob

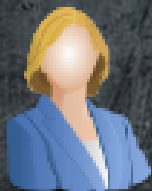


W =

+34687399815
+34671374657
+34759421743

# Protocolo PSI (2 partes)

Alice



V =

+34633946745
+34759421743
+34623434477
+34602305532

Información común: función hash  $H$  y  $p$

$H = \text{SHA512}$

Bob



W =

+34687399815
+34671374657
+34759421743



# Protocolo PSI (2 partes)

Alice



V =

+34633946745
+34759421743
+34623434477
+34602305532

Información común: función hash  $H$  y  $p$

$H = \text{SHA512}$

$p = 4595177610 \dots 3244871619$  (3072 bits)

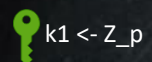
Bob



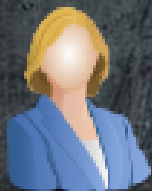
W =

+34687399815
+34671374657
+34759421743

# Protocolo PSI (2 partes)

 $k1 \leftarrow \mathbb{Z}_p$  $d = k1^{-1} \bmod p-1$ 

Alice



V =

+34633946745
+34759421743
+34623434477
+34602305532

Información común: función hash H y p

H = SHA512

p = 4595177610...3244871619 (3072 bits)

Bob

 $k2 \leftarrow \mathbb{Z}_p$ 

W =

+34687399815
+34671374657
+34759421743

REVISAR NO

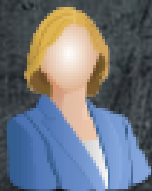


# Protocolo PSI (2 partes)

🔑  $k_1 = 1541338567...2226190981$

Alice

🔑  $d = 3029783844...1310138125$



V =

+34633946745  
+34759421743  
+34623434477  
+34602305532

Información común: función hash H y p

H = SHA512

p = 4595177610...3244871619 (3072 bits)

Bob

🔑  $k_2 = 2177727835...2099723501$



W =

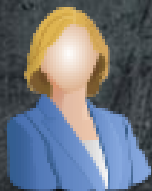
+34687399815  
+34671374657  
+34759421743

# Protocolo PSI (2 partes)

  $k_1 = 1541338567...2226190981$

Alice

  $d = 3029783844...1310138125$



$V =$

+34633946745  
+34759421743  
+34623434477  
+34602305532



$A = H(V)^{k_1} \bmod p =$




Información común: función hash  $H$  y  $p$

$H = \text{SHA512}$

$p = 4595177610...3244871619$  (3072 bits)

Bob

  $k_2 = 2177727835...2099723501$

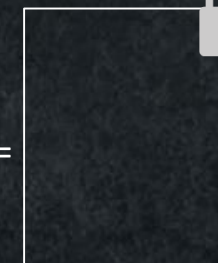


$W =$

+34687399815  
+34671374657  
+34759421743



$B = H(W)^{k_2} \bmod p =$



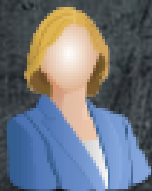


# Protocolo PSI (2 partes)

  $k_1 = 1541338567...2226190981$

Alice

  $d = 3029783844...1310138125$



Información común: función hash  $H$  y  $p$

$H = \text{SHA512}$

$p = 4595177610...3244871619$  (3072 bits)

Bob

  $k_2 = 2177727835...2099723501$



$V =$

+34633946745  
+34759421743  
+34623434477  
+34602305532

$A = H(V)^{k_1} \bmod p =$

39764...9837  
36324...56865  
44736...57169  
41475...46061



$W =$

+34687399815  
+34671374657  
+34759421743

$B = H(W)^{k_2} \bmod p =$

14802...41329  
39967...93643  
86296...43451

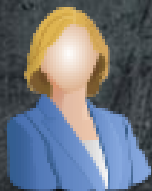


# Protocolo PSI (2 partes)

🔑  $k_1 = 1541338567...2226190981$

Alice

🔑  $d = 3029783844...1310138125$



Información común: función hash  $H$  y  $p$

$H = \text{SHA512}$

$p = 4595177610...3244871619$  (3072 bits)

Bob

🔑  $k_2 = 2177727835...2099723501$



$V =$

+34633946745  
+34759421743  
+34623434477  
+34602305532

$W =$

+34687399815  
+34671374657  
+34759421743

$A = H(V)^{k_1} \bmod p =$

39764...9837  
36324...56865  
44736...57169  
41475...46061

$B = H(W)^{k_2} \bmod p =$

14802...41329  
39967...93643  
86296...43451

$A$

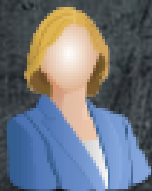


# Protocolo PSI (2 partes)

  $k_1 = 1541338567...2226190981$

Alice

  $d = 3029783844...1310138125$



Información común: función hash  $H$  y  $p$

$H = \text{SHA512}$

$p = 4595177610...3244871619$  (3072 bits)

Bob

  $k_2 = 2177727835...2099723501$



$V =$

+34633946745  
+34759421743  
+34623434477  
+34602305532

$A = H(V)^{k_1} \bmod p =$

39764...9837  
36324...56865  
44736...57169  
41475...46061



$A$

$W =$

+34687399815  
+34671374657  
+34759421743

$B = H(W)^{k_2} \bmod p =$

14802...41329  
39967...93643  
86296...43451



$Z = A^{k_2} \bmod p =$

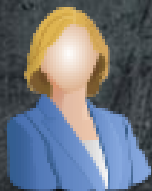


# Protocolo PSI (2 partes)

  $k_1 = 1541338567...2226190981$

Alice

  $d = 3029783844...1310138125$




Información común: función hash  $H$  y  $p$

$H = \text{SHA512}$

$p = 4595177610...3244871619$  (3072 bits)

Bob

  $k_2 = 2177727835...2099723501$



$V =$

+34633946745  
+34759421743  
+34623434477  
+34602305532

$A = H(V)^{k_1} \bmod p =$

39764...9837  
36324...56865  
44736...57169  
41475...46061

$A$

$W =$

+34687399815  
+34671374657  
+34759421743

$B = H(W)^{k_2} \bmod p =$

14802...41329  
39967...93643  
86296...43451

$Z = A^{k_2} \bmod p =$

11913...72722  
14637...99730  
20385...72621  
45301...59046

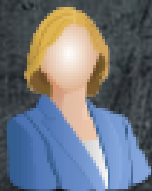


# Protocolo PSI (2 partes)

  $k_1 = 1541338567...2226190981$

Alice

  $d = 3029783844...1310138125$



Información común: función hash  $H$  y  $p$

$H = \text{SHA512}$

$p = 4595177610...3244871619$  (3072 bits)

Bob

  $k_2 = 2177727835...2099723501$



$V =$

+34633946745  
+34759421743  
+34623434477  
+34602305532

$A = H(V)^{k_1} \bmod p =$

39764...9837  
36324...56865  
44736...57169  
41475...46061

$W =$

+34687399815  
+34671374657  
+34759421743

$B = H(W)^{k_2} \bmod p =$

14802...41329  
39967...93643  
86296...43451

$A$

$Z, B$

$Z = A^{k_2} \bmod p =$

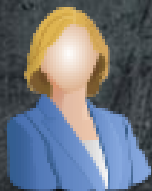
11913...72722  
14637...99730  
20385...72621  
45301...59046

# Protocolo PSI (2 partes)

  $k_1 = 1541338567...2226190981$

Alice

  $d = 3029783844...1310138125$




Información común: función hash  $H$  y  $p$

$H = \text{SHA512}$

$p = 4595177610...3244871619$  (3072 bits)

Bob

  $k_2 = 2177727835...2099723501$



$V =$

+34633946745  
+34759421743  
+34623434477  
+34602305532

$A = H(V)^{k_1} \bmod p =$

39764...9837  
36324...56865  
44736...57169  
41475...46061

$\text{dec} = Z^d \bmod p =$

$W =$

+34687399815  
+34671374657  
+34759421743

$B = H(W)^{k_2} \bmod p =$

14802...41329  
39967...93643  
86296...43451

$Z = A^{k_2} \bmod p =$

11913...72722  
14637...99730  
20385...72621  
45301...59046

$A$

$Z, B$

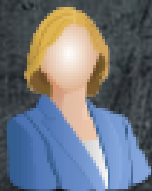


# Protocolo PSI (2 partes)

🔑  $k_1 = 1541338567...2226190981$

Alice

🔑  $d = 3029783844...1310138125$



Información común: función hash  $H$  y  $p$

$H = \text{SHA512}$

$p = 4595177610...3244871619$  (3072 bits)

Bob

🔑  $k_2 = 2177727835...2099723501$



$V =$

+34633946745  
+34759421743  
+34623434477  
+34602305532

$W =$

+34687399815  
+34671374657  
+34759421743

$A = H(V)^{k_1} \bmod p =$

39764...9837  
36324...56865  
44736...57169  
41475...46061

$B = H(W)^{k_2} \bmod p =$

14802...41329  
39967...93643  
86296...43451

$A$

$Z, B$

$\text{dec} = Z^d \bmod p =$

37784...4843  
86296...43451  
11955...12118  
36505...25112

$Z = A^{k_2} \bmod p =$

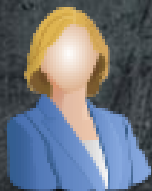
11913...72722  
14637...99730  
20385...72621  
45301...59046

# Protocolo PSI (2 partes)

  $k_1 = 1541338567...2226190981$

Alice

  $d = 3029783844...1310138125$



Información común: función hash  $H$  y  $p$

$H = \text{SHA512}$

$p = 4595177610...3244871619$  (3072 bits)

Bob

  $k_2 = 2177727835...2099723501$



$V =$

+34633946745  
+34759421743  
+34623434477  
+34602305532

$A = H(V)^{k_1} \bmod p =$

39764...9837  
36324...56865  
44736...57169  
41475...46061

$W =$

+34687399815  
+34671374657  
+34759421743

$B = H(W)^{k_2} \bmod p =$

14802...41329  
39967...93643  
86296...43451

$A$

$Z, B$

$\text{dec} = Z^d \bmod p =$

37784...4843  
86296...43451  
11955...12118  
36505...25112

$Z = A^{k_2} \bmod p =$

11913...72722  
14637...99730  
20385...72621  
45301...59046

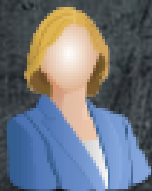


# Protocolo PSI (2 partes)

  $k_1 = 1541338567...2226190981$

Alice

  $d = 3029783844...1310138125$




Información común: función hash  $H$  y  $p$

$H = \text{SHA512}$

$p = 4595177610...3244871619$  (3072 bits)

Bob

  $k_2 = 2177727835...2099723501$



$V =$

+34633946745  
+34759421743  
+34623434477  
+34602305532

$A = H(V)^{k_1} \bmod p =$

39764...9837  
36324...56865  
44736...57169  
41475...46061

$\text{dec} = Z^d \bmod p =$

37784...4843  
86296...43451  
11955...12118  
36505...25112

$W =$

+34687399815  
+34671374657  
+34759421743

$B = H(W)^{k_2} \bmod p =$

14802...41329  
39967...93643  
86296...43451

$Z = A^{k_2} \bmod p =$

11913...72722  
14637...99730  
20385...72621  
45301...59046

$A$

$Z, B$

Poner aquí abajo desglosado que tiene Alice

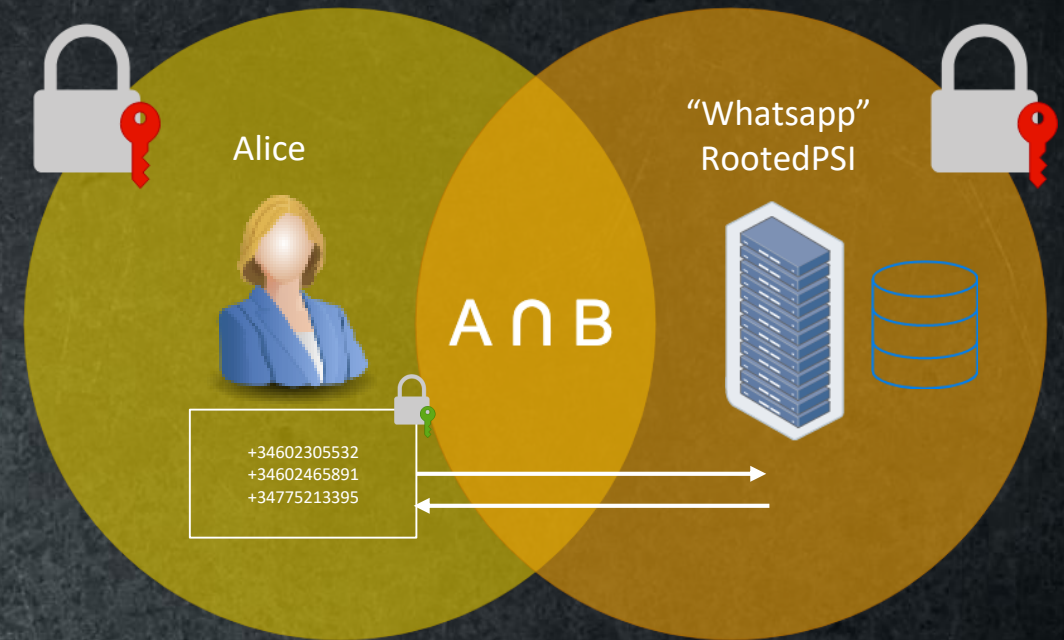
Desglosar  $B$ ,  $Z$  y el cálculo  $Z^d \bmod p$ !!!!

Destacar que la comprobación la hace  $A$ , que pasaría si  $B$  llena toda su agenda de contactos?



# Protocolo PSI (demo)

- Aplicación que implementa el **protocolo PSI** entre un cliente y un servidor.
- Obtiene los **contactos de la agenda del cliente** que están ya en la aplicación de mensajería.
- Ver [Kales2019] para un ejemplo de **protocolo escalable**.







# Protocolo PSI (demo)

## RootedPSI


### My contacts




Elena Gascón  
📞 0034133946745




Óscar Tejera  
📞 0034459421743




Margarita Guerra  
📞 0034323434477




Nicolás Calzada  
📞 0034102305532




Mónica Caballero  
📞 0034694357181



Vanesa Múñiz  
📞 0034230868574



Sandra Matas  
📞 0034434018353



Luís Vilanova  
📞 0034004569373

Check common contacts

Elements Console Sources Network Performance Memory Application Security Audits

top Filter Default levels

Generating prime number...  
Generated prime number: ca7c7c6aecb70653600...4d9fd8779abcd943d3c3  
Generating client key...  
Generated client key: b532c748fb6daefac376...386fc4b0cca58df96aad  
Hashing and encrypting phones...

app.7c517a20.js:1  
app.7c517a20.js:1  
app.7c517a20.js:1  
app.7c517a20.js:1  
app.7c517a20.js:1

53

# Protocolo Multiparty PSI (MPSI)

- Permite el cómputo de los elementos comunes entre **n partes**.
- Requiere una parte adicional denominada “**Dealer**” (D) => p.e un servidor
- Basado en que la hipótesis **DDH** es **difícil computacionalmente**.
- Seguro contra adversarios **semihonestos**.
- Requiere de **filtros de Bloom** [Bloom1970] y cifrado **ElGamal exponencial** distribuido (o su variante con curvas elípticas).



# Explicación - Filtros de Bloom

- Permite comprobar si un **elemento pertenece a un conjunto**.
- **Vector de bits** que representa un conjunto.
- Usa un número fijado de **funciones hash** (no necesariamente criptográficas).
- Permite **falsos positivos**, pero **no los falsos negativos** (si no está en el grupo no lo está).
- Elementos pueden ser **añadidos**, pero **no eliminarlos** del filtro.

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

# Explicación - Filtros de Bloom

- Permite comprobar si un elemento pertenece a un conjunto.
- **Vector de bits** que representa un conjunto.
- Usa un número fijado de **funciones hash** (no necesariamente criptográficas).
- Permite **falsos positivos**, pero **no los falsos negativos**.
- Elementos pueden ser **añadidos**, pero **no eliminarlos** del filtro.

Rooted																	Rooted	
		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	fnn	6
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	murmur	3



# Explicación - Filtros de Bloom

- Permite comprobar si un elemento pertenece a un conjunto.
- **Vector de bits** que representa un conjunto.
- Usa un número fijado de **funciones hash** (no necesariamente criptográficas).
- Permite **falsos positivos**, pero **no los falsos negativos**.
- Elementos pueden ser **añadidos**, pero **no eliminarlos** del filtro.

Rooted	
0	0
0	0
0	1
0	0
0	0
1	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0

Rooted	
fnv	6
murmur	3

# Explicación - Filtros de Bloom

- Permite comprobar si un elemento pertenece a un conjunto.
- **Vector de bits** que representa un conjunto.
- Usa un número fijado de **funciones hash** (no necesariamente criptográficas).
- Permite **falsos positivos**, pero **no los falsos negativos**.
- Elementos pueden ser **añadidos**, pero **no eliminarlos** del filtro.

Valencia																	Valencia	
		0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	fnn	7
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	murmur	2



# Explicación - Filtros de Bloom

- Permite comprobar si un elemento pertenece a un conjunto.
- **Vector de bits** que representa un conjunto.
- Usa un número fijado de **funciones hash** (no necesariamente criptográficas).
- Permite **falsos positivos**, pero **no los falsos negativos**.
- Elementos pueden ser **añadidos**, pero **no eliminarlos** del filtro.

Valencia																	Valencia	
		0	0	1	1	0	0	1	1	0	0	0	0	0	0	0	fnn	7
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	murmur	2

# Explicación - Filtros de Bloom

- Permite comprobar si un **elemento pertenece a un conjunto**.
- **Vector de bits** que representa un conjunto.
- Usa un número fijado de **funciones hash** (no necesariamente criptográficas).
- Permite **falsos positivos**, pero **no los falsos negativos**.
- Elementos pueden ser **añadidos**, pero **no eliminarlos** del filtro.

2019																	2019	
		0	0	1	1	0	0	1	1	0	0	0	0	0	0	0	fnv	2
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	murmur	11



# Explicación - Filtros de Bloom

- Permite comprobar si un **elemento pertenece a un conjunto**.
- **Vector de bits** que representa un conjunto.
- Usa un número fijado de **funciones hash** (no necesariamente criptográficas).
- Permite **falsos positivos**, pero **no los falsos negativos**.
- Elementos pueden ser **añadidos**, pero **no eliminarlos** del filtro.

2019																	2019	
		0	0	1	1	0	0	1	1	0	0	0	1	0	0	0	fnv	2
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	murmur	11

# Explicación - ElGamal exponencial distribuido

- **Generación de clave:**

- $F_p$  es un cuerpo y  $g$  un generador del cuerpo de orden  $q$ .
- Cada parte elige un  $x_i$  al azar de  $Z_q$  y calcula  $y_i = g^{x_i} \bmod p$ .
- La clave pública es  $y = \text{prod}(y_i)$ .

- **Cifrado:**

- $E(m, y; r) = (g^r \bmod p, g^m * y^r \bmod p) = (u, v)$

- **Descifrado:**

- Cada parte calcula  $z_i = u^{x_i} \bmod p$  y  $z = \text{prod}(z_i)$  conjuntamente.
- Se descifra como  $v/z \bmod p = g^m$ .

- **Se cumple que**

- Es un cifrado **homomórfico aditivo**:  $E(m1) * E(m2) = E(m1+m2)$ .
- Es un cifrado **homomórfico por un escalar**:  $E(m)^k = E(k*m)$ .



# Protocollo Multiparty PSI (MPSI)

Alice



+34633946745  
+34759421743  
+34623434477  
+34602305532

Carol



+34759421743  
+34678373889  
+34700636373  
+34678543123



Dealer



Bob

+34687399815  
+34671374657  
+34759421743

# Protocolo Multiparty PSI (MPSI)

Alice



+34633946745  
+34759421743  
+34623434477  
+34602305532

Carol



+34759421743  
+34678373889  
+34700636373  
+34678543123

Información común:  $g, p$



Dealer



Bob

+34687399815  
+34671374657  
+34759421743



# Protocolo Multiparty PSI (MPSI)

Alice



+34633946745  
+34759421743  
+34623434477  
+34602305532

Información común:  $g, p$

$g = 3$

Carol



+34759421743  
+34678373889  
+34700636373  
+34678543123



Dealer



Bob

+34687399815  
+34671374657  
+34759421743

# Protocolo Multiparty PSI (MPSI)

Alice



+34633946745  
+34759421743  
+34623434477  
+34602305532

Información común:  $g, p$

$g = 3$

$p = 0xc7611ad4d3...c94f0a88fb$  (3072 bits)

Carol



+34759421743  
+34678373889  
+34700636373  
+34678543123



Dealer



Bob

+34687399815  
+34671374657  
+34759421743



# Protocolo Multiparty PSI (MPSI)

Alice



+34633946745  
+34759421743  
+34623434477  
+34602305532

Carol



+34759421743  
+34678373889  
+34700636373  
+34678543123



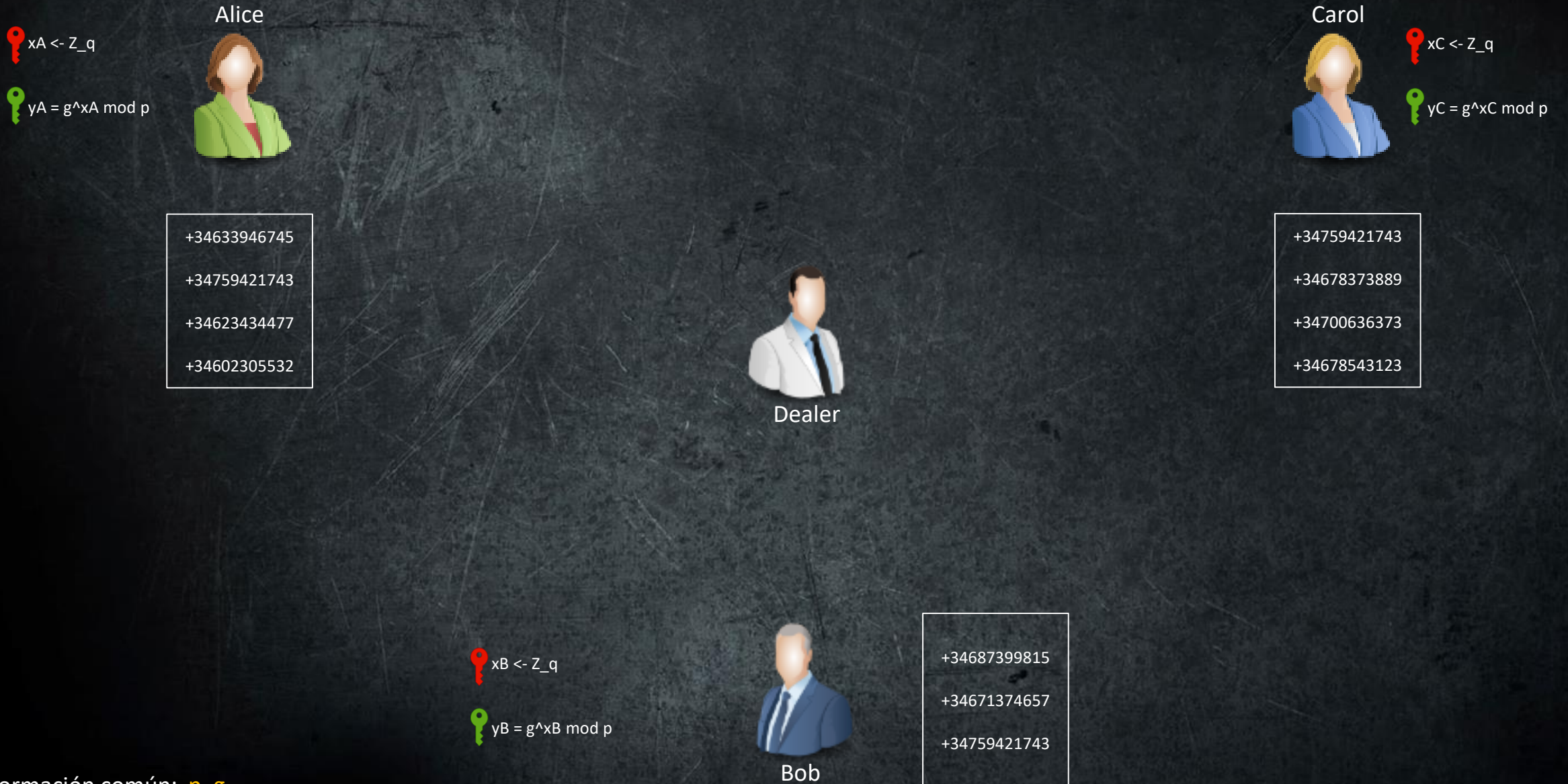
Dealer



Bob

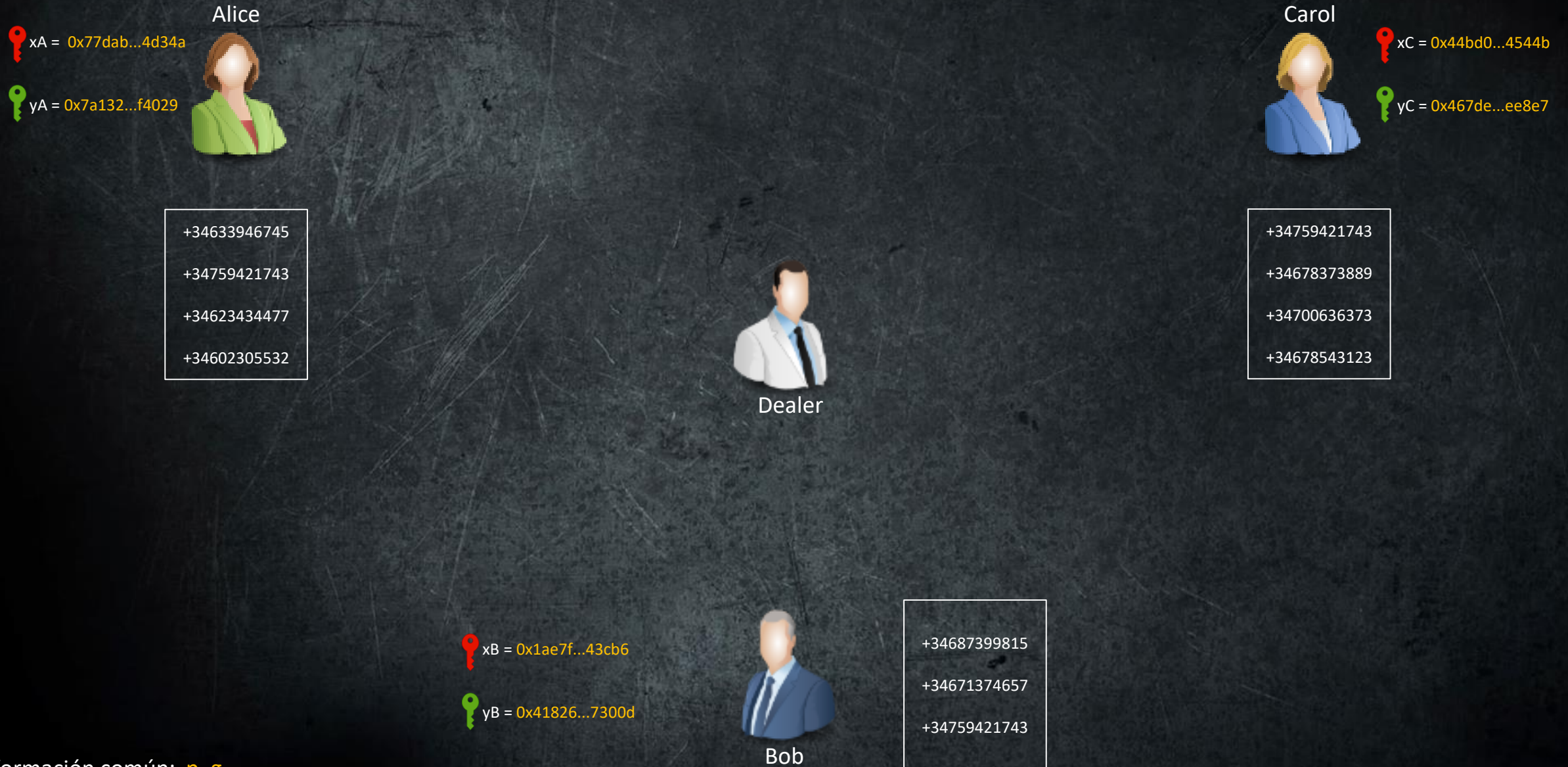
+34687399815  
+34671374657  
+34759421743

# Protocolo Multiparty PSI (MPSI)

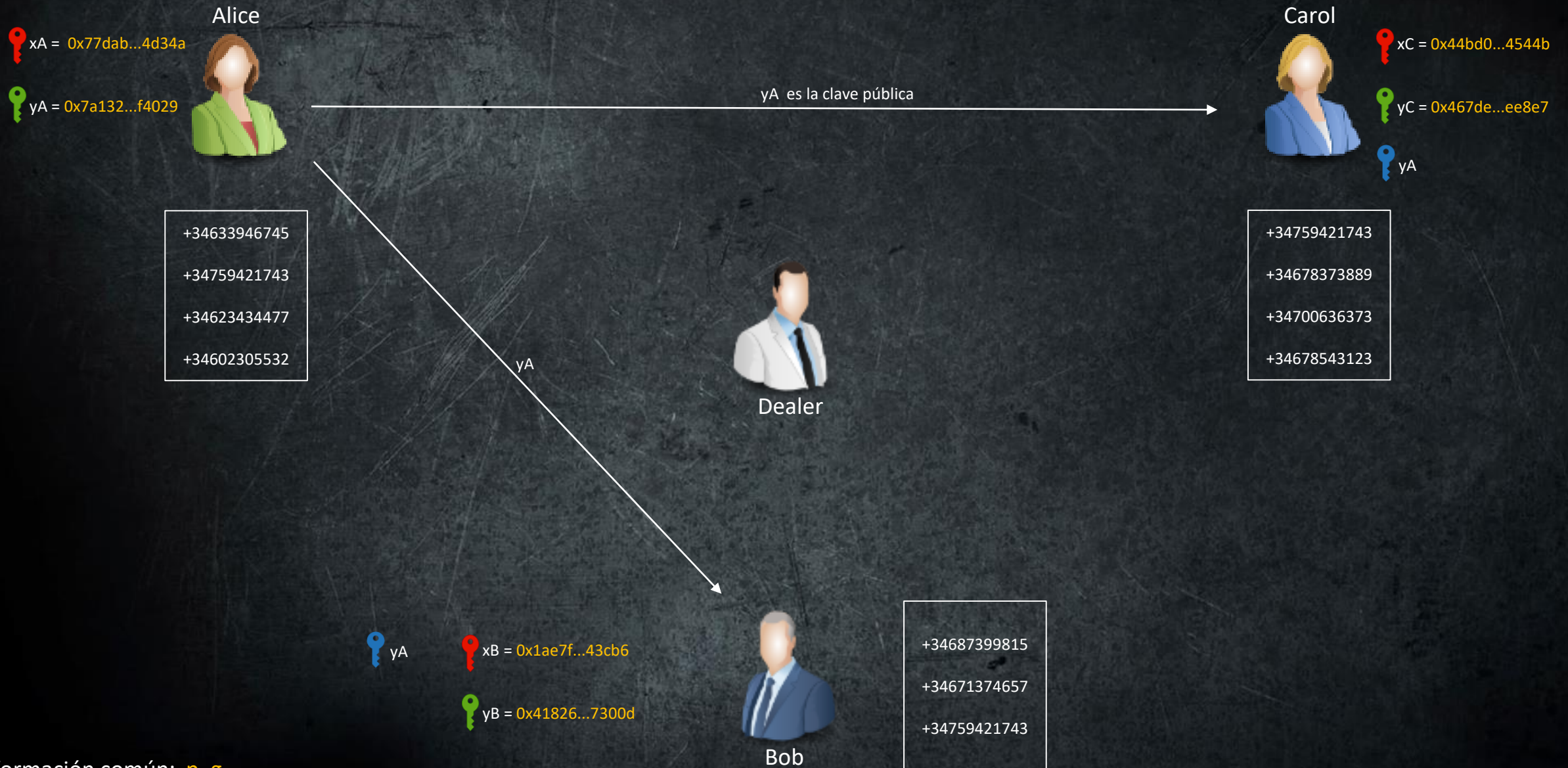




# Protocolo Multiparty PSI (MPSI)

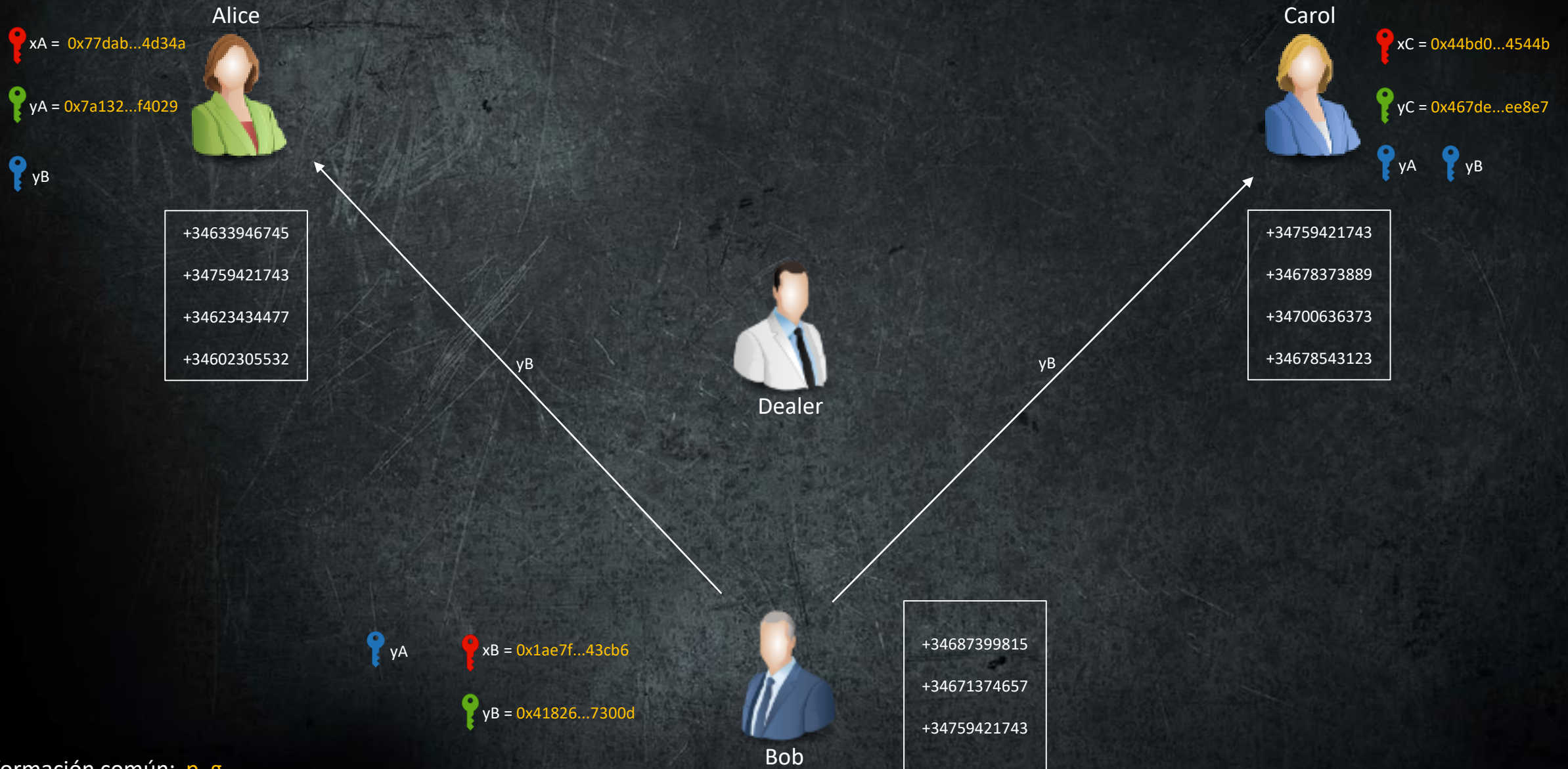


# Protocolo Multiparty PSI (MPSI)

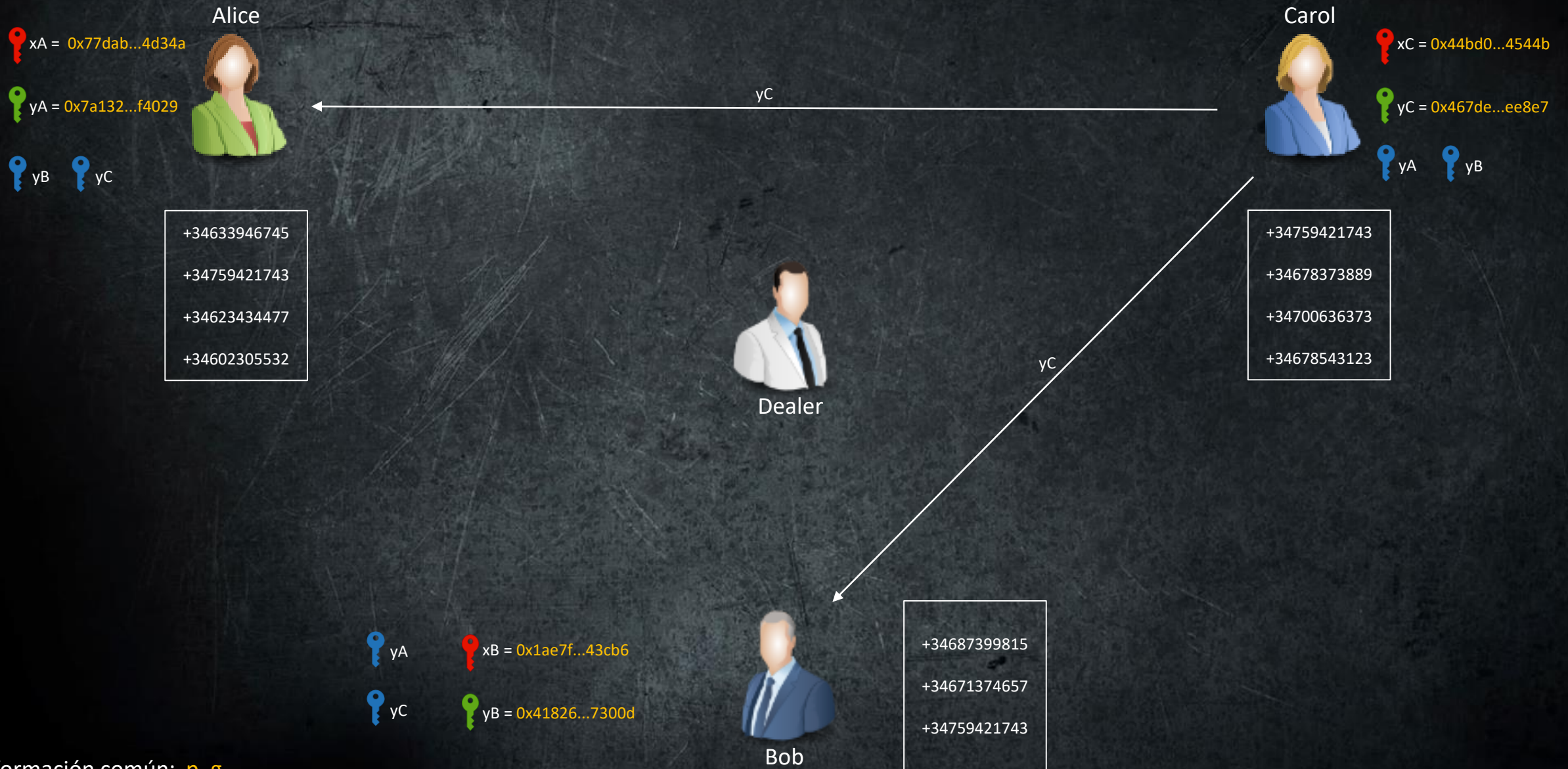




# Protocolo Multiparty PSI (MPSI)



# Protocolo Multiparty PSI (MPSI)





# Protocolo Multiparty PSI (MPSI)

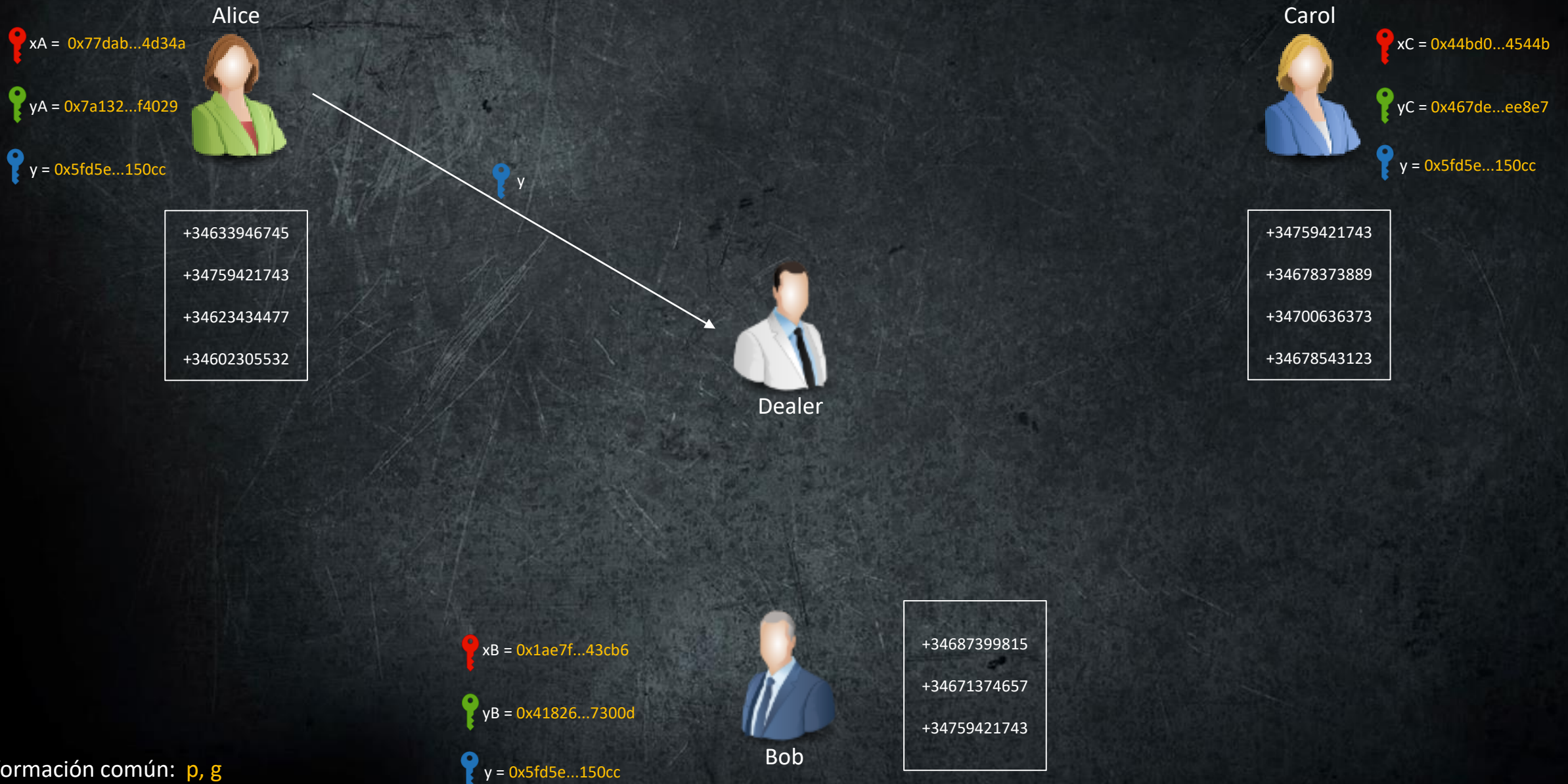


# Protocolo Multiparty PSI (MPSI)





# Protocolo Multiparty PSI (MPSI)



# Protocollo Multiparty PSI (MPSI)

xA = 0x77dab...4d34a



Key:  $y_A = 0x7a132\dots f4029$

key  $y = 0x5fd5e...150cc$

+34633946745  
+34759421743  
+34623434477  
+34602305532

xC = 0x44bd0...4544b



Key: `yC = 0x467de...ee8e7`

🔑  $y = 0x5fd5e...150cc$

+34759421743  
+34678373889  
+34700636373  
+34678543123



## Dealer

xB = 0x1ae7f...43cb6



 yB = 0x41826...7300d

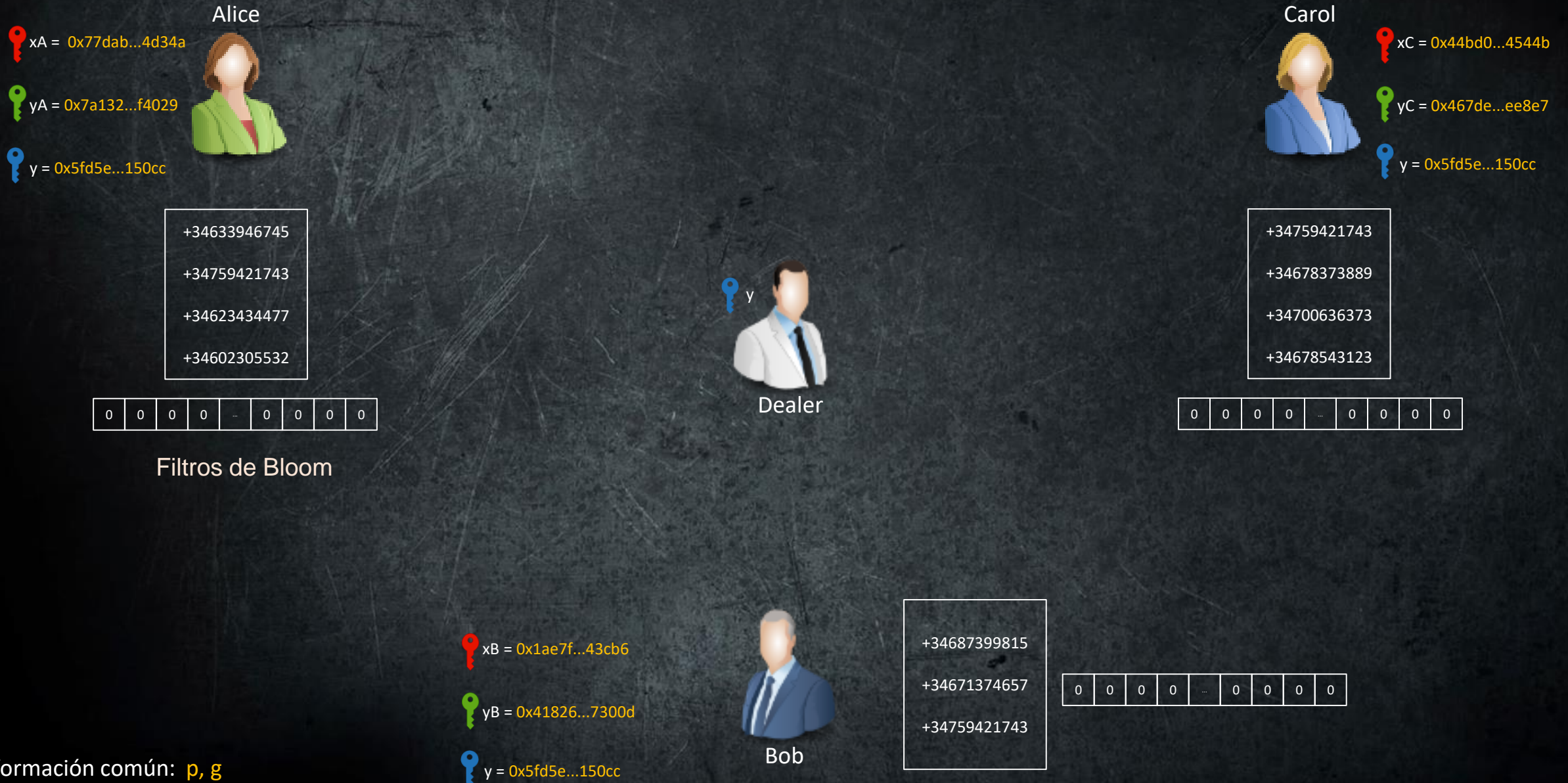
Key  $y = 0x5fd5e...150cc$

+34687399815  
+34671374657  
+34759421743

Información común:  $p, g$



# Protocolo Multiparty PSI (MPSI)



# Protocolo Multiparty PSI (MPSI)





# Protocolo Multiparty PSI (MPSI)

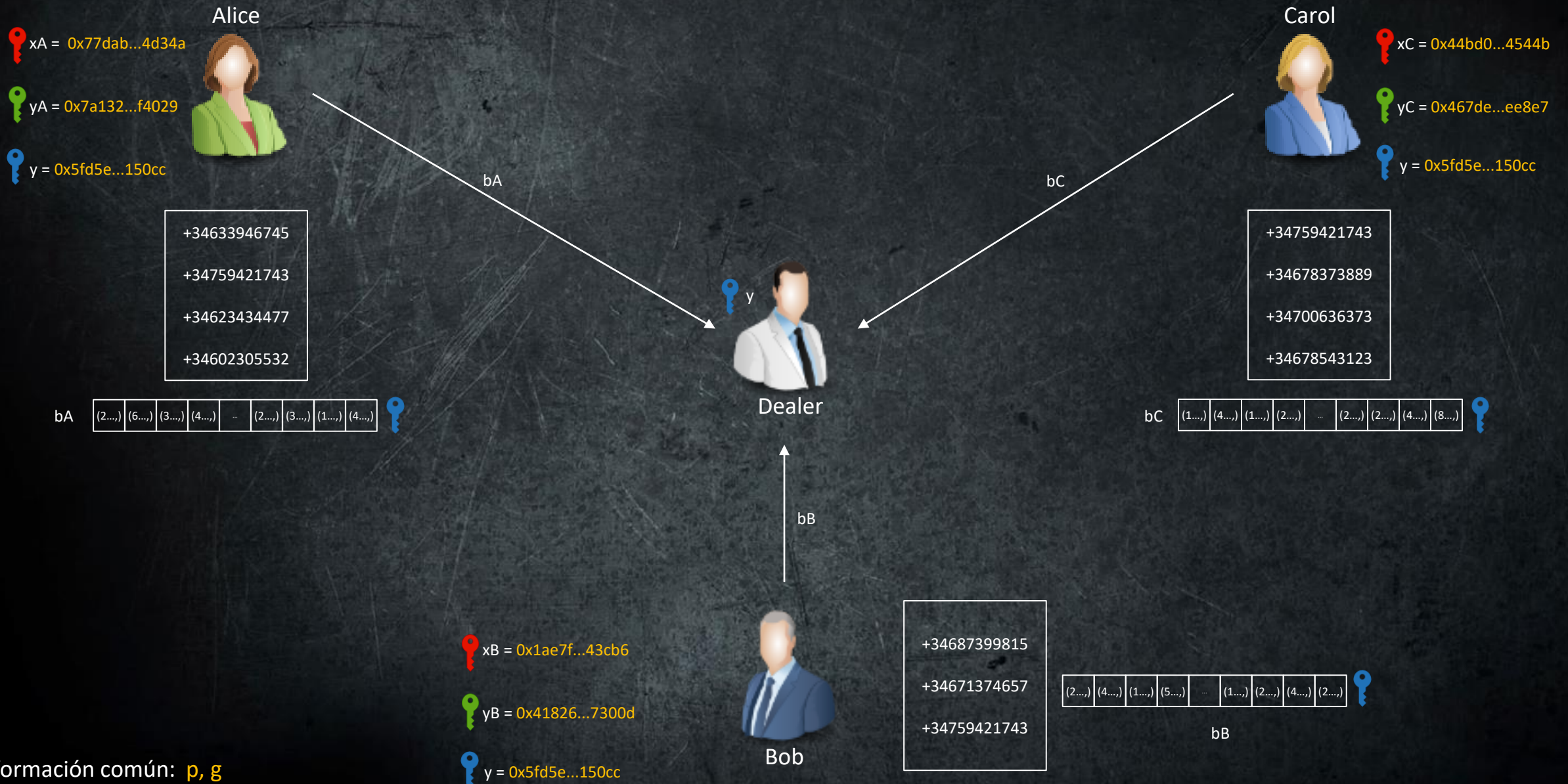


# Protocolo Multiparty PSI (MPSI)

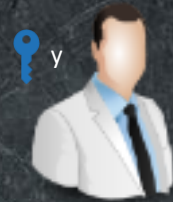







# Protocolo Multiparty PSI (MPSI)



# Protocolo Multiparty PSI (MPSI)

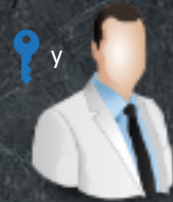


Dealer

bA	(2...)	(6...)	(3...)	(4...)	...	(2...)	(3...)	(1...)	(4...)	
bB	(2...)	(4...)	(1...)	(5...)	...	(1...)	(2...)	(4...)	(2...)	
bC	(1...)	(4...)	(1...)	(2...)	...	(2...)	(2...)	(4...)	(8...)	



# Protocolo Multiparty PSI (MPSI)



Dealer

bA 

(2...)	(6...)	(3...)	(4...)	...	(2...)	(3...)	(1...)	(4...)
--------	--------	--------	--------	-----	--------	--------	--------	--------



bB 

(2...)	(4...)	(1...)	(5...)	...	(1...)	(2...)	(4...)	(2...)
--------	--------	--------	--------	-----	--------	--------	--------	--------



bC 


(1...)	(4...)	(1...)	(2...)	...	(2...)	(2...)	(4...)	(8...)
--------	--------	--------	--------	-----	--------	--------	--------	--------



---

bA \* bB \* bC 

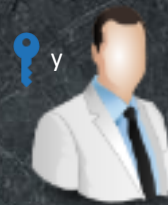
(3...)	(5...)	(5...)	(4...)	...	(8...)	(3...)	(2...)	(1...)
--------	--------	--------	--------	-----	--------	--------	--------	--------



# Protocolo Multiparty PSI (MPSI)

-n

-3	-3	-3	-3	...	-3	-3	-3	-3
----	----	----	----	-----	----	----	----	----



Dealer

bA

(2...)	(6...)	(3...)	(4...)	...	(2...)	(3...)	(1...)	(4...)
--------	--------	--------	--------	-----	--------	--------	--------	--------



bB

(2...)	(4...)	(1...)	(5...)	...	(1...)	(2...)	(4...)	(2...)
--------	--------	--------	--------	-----	--------	--------	--------	--------



bC


(1...)	(4...)	(1...)	(2...)	...	(2...)	(2...)	(4...)	(8...)
--------	--------	--------	--------	-----	--------	--------	--------	--------




---

bA \* bB \* bC

(3...)	(5...)	(5...)	(4...)	...	(8...)	(3...)	(2...)	(1...)
--------	--------	--------	--------	-----	--------	--------	--------	--------

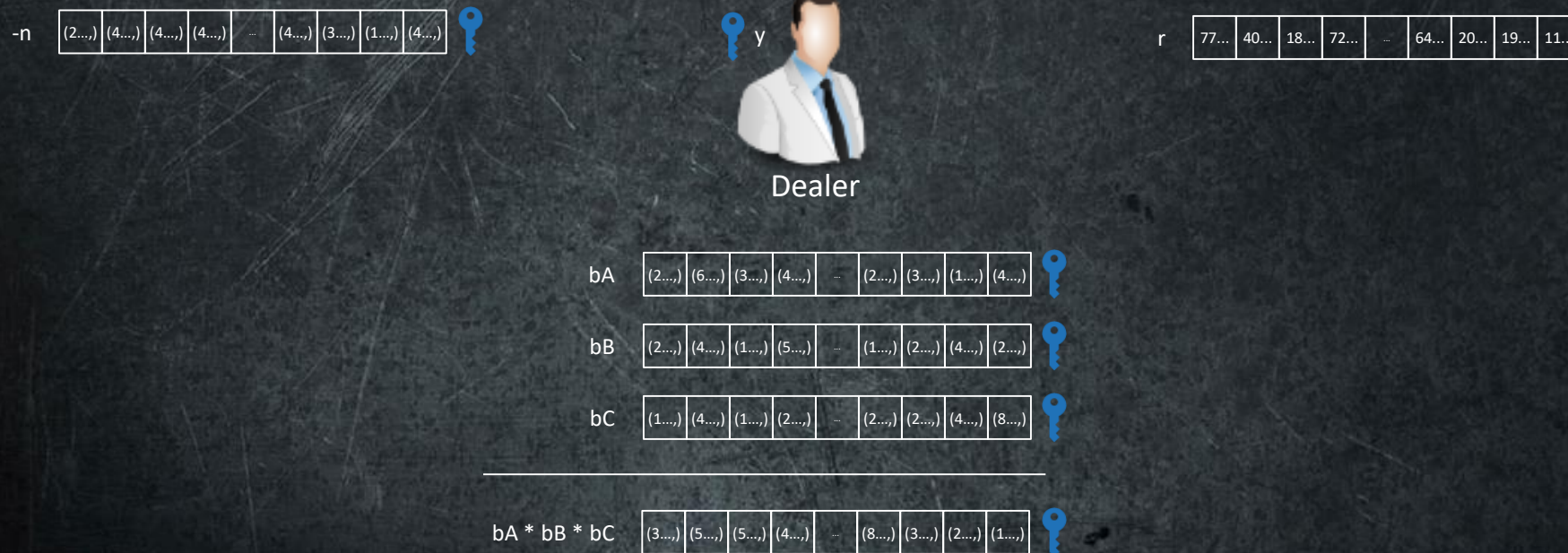




# Protocolo Multiparty PSI (MPSI)

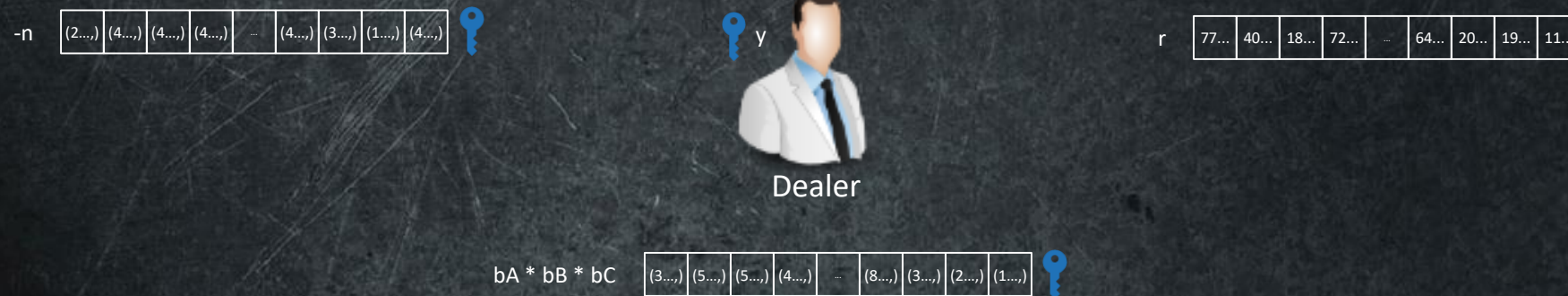


# Protocolo Multiparty PSI (MPSI)

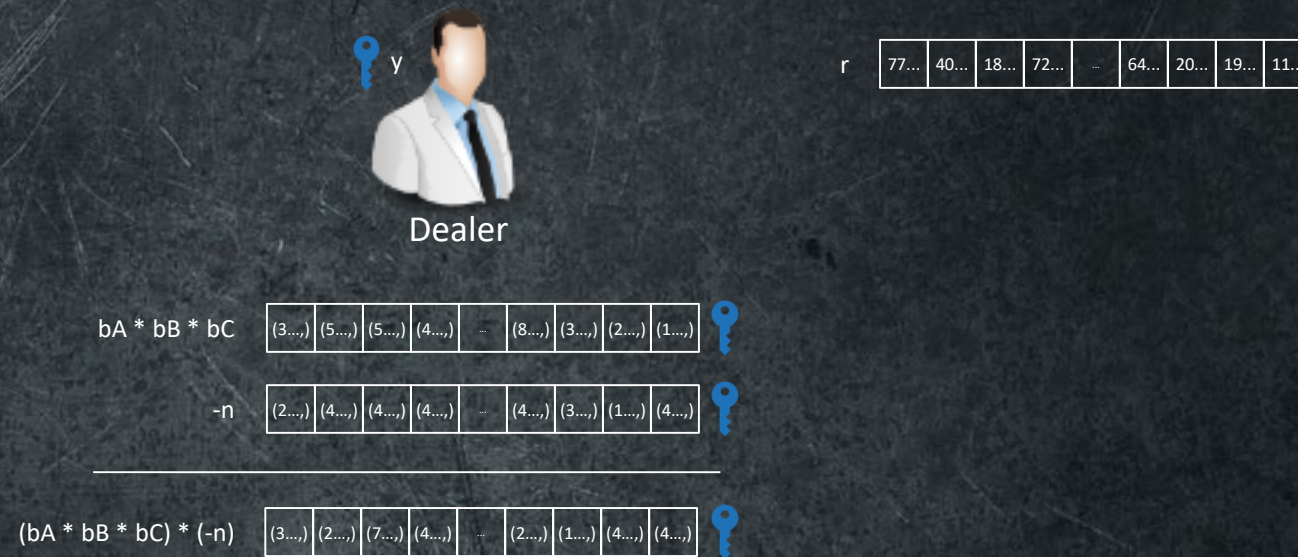




# Protocolo Multiparty PSI (MPSI)

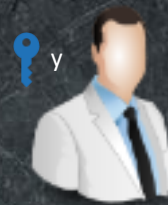


# Protocolo Multiparty PSI (MPSI)





# Protocolo Multiparty PSI (MPSI)



Dealer

 $(bA * bB * bC) * (-n)$ 

(3...)	(2...)	(7...)	(4...)	...	(2...)	(1...)	(4...)	(4...)
--------	--------	--------	--------	-----	--------	--------	--------	--------



r

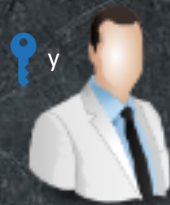
77...	40...	18...	72...	...	64...	20...	19...	11...
-------	-------	-------	-------	-----	-------	-------	-------	-------

 $((bA * bB * bC) * (-n))^r$ 

(2...)	(4...)	(3...)	(2...)	...	(3...)	(1...)	(2...)	(2...)
--------	--------	--------	--------	-----	--------	--------	--------	--------



# Protocolo Multiparty PSI (MPSI)



Dealer

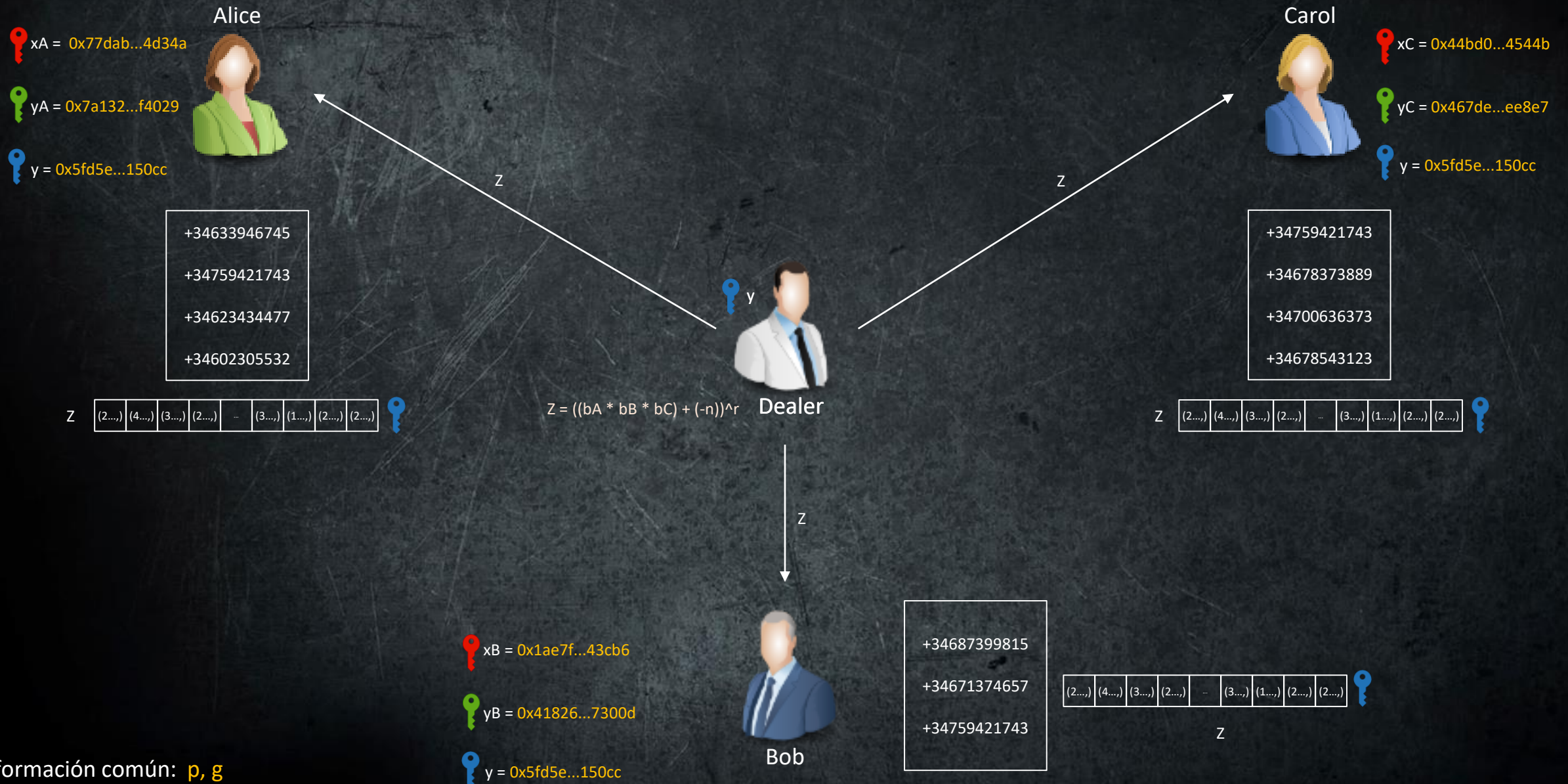
$$Z = ((bA * bB * bC) + (-n))^r$$

(2,...)	(4,...)	(3,...)	(2,...)	...	(3,...)	(1,...)	(2,...)	(2,...)
---------	---------	---------	---------	-----	---------	---------	---------	---------







# Protocolo Multiparty PSI (MPSI)





# Protocolo Multiparty PSI (MPSI)

Alice

  $x_A = 0x77dab...4d34a$

  $y_A = 0x7a132...f4029$

  $y = 0x5fd5e...150cc$





+34633946745
+34759421743
+34623434477
+34602305532


dec


1	11...	31...	77...	...	41...	18...	55...	10...
---	-------	-------	-------	-----	-------	-------	-------	-------

Carol

  $x_C = 0x44bd0...4544b$

  $y_C = 0x467de...ee8e7$


  $y = 0x5fd5e...150cc$





+34759421743
+34678373889
+34700636373
+34678543123


dec

1	11...	31...	77...	...	41...	18...	55...	10...
---	-------	-------	-------	-----	-------	-------	-------	-------

  $x_B = 0x1ae7f...43cb6$

  $y_B = 0x41826...7300d$

  $y = 0x5fd5e...150cc$



+34687399815
+34671374657
+34759421743




1	11...	31...	77...	...	41...	18...	55...	10...
---	-------	-------	-------	-----	-------	-------	-------	-------


dec



# Protocolo Multiparty PSI (MPSI)

Alice

  $x_A = 0x77dab...4d34a$   
  $y_A = 0x7a132...f4029$   
  $y = 0x5fd5e...150cc$







+34633946745  
+34759421743  
+34623434477  
+34602305532

dec 

1	0	0	0	...	0	0	0	0
---	---	---	---	-----	---	---	---	---

Carol




  $x_C = 0x44bd0...4544b$   
  $y_C = 0x467de...ee8e7$   
  $y = 0x5fd5e...150cc$



+34759421743  
+34678373889  
+34700636373  
+34678543123

dec 

1	0	0	0	...	0	0	0	0
---	---	---	---	-----	---	---	---	---

  $x_B = 0x1ae7f...43cb6$   
  $y_B = 0x41826...7300d$   
  $y = 0x5fd5e...150cc$



Bob


+34687399815  
+34671374657  
+34759421743


1	0	0	0	...	0	0	0	0
---	---	---	---	-----	---	---	---	---


dec


# Protocolo Multiparty PSI (MPSI)

Alice

  $x_A = 0x77dab...4d34a$

  $y_A = 0x7a132...f4029$

  $y = 0x5fd5e...150cc$





+34633946745  
+34759421743  
+34623434477  
+34602305532


dec 


1	0	0	0	...	0	0	0	0
---	---	---	---	-----	---	---	---	---

Carol

  $x_C = 0x44bd0...4544b$

  $y_C = 0x467de...ee8e7$


  $y = 0x5fd5e...150cc$





+34759421743  
+34678373889  
+34700636373  
+34678543123

dec 

1	0	0	0	...	0	0	0	0
---	---	---	---	-----	---	---	---	---

  $x_B = 0x1ae7f...43cb6$

  $y_B = 0x41826...7300d$

  $y = 0x5fd5e...150cc$



Bob

+34687399815  
+34671374657  
+34759421743

1	0	0	0	...	0	0	0	0
---	---	---	---	-----	---	---	---	---

dec

Información común:  $p, g$



# Protocolo MPSI (demo)

## Common parameters

Parameter	Value
Curve	P-256
Generator	('0x6b17d1f2e12c4247f8bce6e563a440f277037d812deb33a0f4a13945d898c296', '0x4fe342e2fe1a7f9b8ee7eb4a7c0f9e162bce33576b315ececbb6406837bf51f5')
Order	0xffffffff00000000ffffffffffffffffbce6faada7179e84f3b9cac2fc632551
# parties	3
Max elements (BF)	100
Error rate (BF)	0.00001

## Information about n parties

Party	Secret key	Public key	Data	Data
1	0x74adb788c3b04f0ba9d985e562ece538cca23d77461fc0563273618a38768b6	('0x7e8e2682b42f953c3687fa311c04c0514b36538c19903274836c396856b621b75', '0xaad356eb24d9c64a5e4556437abb22573c6783358a51def1e2cb382bb447b23')	99	['0034133946745', '0034459421743', '0034323434477', '0034327331103', '0034694357181', '0034563832690', '0034747696009', '0034418535596', '0034715637304', '0034808883521', '0034624628375', '0034129702161', '0034004569373', '0034547708506', '0034322276792', '0034305501112', '0034708854021', '0034178995261', '0034629718764', '0034465370359', '0034672152881', '0034091255731', '0034366962815', '0034580844594', '0034703895360', '0034374038759', '0034120622891', '0034053929528', '0034209108629', '0034624487369', '0034805660466', '0034787464563', '0034454708764', '0034313656693', '0034509459854', '0034487165768', '0034189580399', '0034980142187', '0034972263833', '0034285097179', '0034636151176', '0034543220397', '0034813088233', '0034996820497', '0034016464901', '0034271780088', '0034704982616', '0034866059862', '0034109023738', '0034216882845', '0034662650352', '0034199047996', '0034212531028', '0034388009181', '0034086401907', '0034384693668', '0034893018553', '0034477712320', '0034229910017', '0034486004670', '0034731601753', '0034690023529', '0034291956650', '0034147790474', '0034285982999', '0034773985659', '0034185936256', '0034791052114', '0034665670501',

# Reflexión: Intercambio de agendas

- Whatsapp, Telegram, Signal, Threema, Riot, ...

Comparison	Allo	iMessage	Messenger	Riot	Signal	Skype	Telegram	Threema	Viber	Whatsapp	Wickr	Wire		
Is encryption turned on by default?	No	Yes	No	No	Yes	Yes	No	Yes	Yes (if device supports it)	Yes (if device supports it)	Yes	Yes	Yes	Yes
Cryptographic primitives		RSA-1280 (encryption), ECDSA 256 (signing) / AES 128 / SHA-1	Curve25519 / AES-256 / HMAC-SHA256	Curve25519 / AES-256 / HMAC-SHA256	Curve25519 / AES-256 / HMAC-SHA256	RSA-1536 & 2048 / AES 256 / SHA-1	RSA 2048 / AES 256 / SHA-256	Curve25519 256 / XSalsa20 256 / Poly1305-AES 128	Curve25519 256 / Salsa20 128 / HMAC-SHA256	Curve25519 / AES-256 / HMAC-SHA256	ECDH512 / AES-256 / HMAC-SHA256	Curve25519 / ChaCha20 / HMAC-SHA256		
Are the app and server completely open source?	No	No	No	Yes	Yes	No	No (clients and API only)	No	No	No	No	No	Yes	
Can you sign up to the app anonymously?	No	No	No	Yes	No	No	No	Yes	No	No	Yes	No		
Can you add a contact without needing to trust a directory server?	No	No	No	No	No	No	No	Yes	Yes	No	No	No		
Can you manually verify contacts' fingerprints?	No	No	Yes	Yes	Yes	No	No (session only, does not provide users' fingerprint information)	Yes	Yes	Yes	Yes	Yes		
Directory service could be modified to enable a MITM attack?	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes		



# “Apostando fuerte” – Uniendo partes...



# Private Intersection-Sum (PIS)

- Permite calcular la **intersección de 2 conjuntos y sumar** valores enteros asociados a los conjuntos, sin revelar información sobre sus datos.
- Basado en que la hipótesis **DDH** es **computacionalmente difícil**.
- Utiliza un protocolo **PSI** y **PHE** (ElGamal con curvas elípticas).
- Seguro contra adversarios **semihonestos**.
- Requiere del **cifrado de exponenciación de Pohlig-Hellman** [Holden2008, Pohlig1978].
- Desarrollado por Google y código disponible en GitHub:  
<https://github.com/google/private-join-and-compute>



# Private Intersection-Sum (PIS)

Alice

 $V =$ 

+34633946745
+34759421743
+34623434477
+34602305532

Bob

 $W = (w_j, t_j) =$ 

+34687399815	2
+34623434477	9
+34759421743	10

# Private Intersection-Sum (PIS)

Alice

 $V =$ 

+34633946745
+34759421743
+34623434477
+34602305532

Información común: función hash  $H$  y  $p$

Bob

 $W = (w_j, t_j) =$ 

+34687399815	2
+34623434477	9
+34759421743	10



# Private Intersection-Sum (PIS)

Alice

 $V =$ 

+34633946745
+34759421743
+34623434477
+34602305532

Información común: función hash  $H$  y  $p$

 $H = \text{SHA512}$ 

Bob

 $W = (w_j, t_j) =$ 

+34687399815	2
+34623434477	9
+34759421743	10

# Private Intersection-Sum (PIS)

Alice

 $V =$ 

+34633946745
+34759421743
+34623434477
+34602305532

Información común: función hash  $H$  y  $p$

 $H = \text{SHA512}$  $p = 5573842157 \dots 6869429667$  (3072 bits)

Bob

 $W = (w_j, t_j) =$ 

+34687399815	2
+34623434477	9
+34759421743	10



# Private Intersection-Sum (PIS)

🔑  $k_1 = 3435215506...4956620898$

Alice



$V =$

+34633946745
+34759421743
+34623434477
+34602305532

Información común: función hash  $H$  y  $p$

$H = \text{SHA512}$

$p = 5573842157...6869429667$  (3072 bits)

Bob

🔑  $k_2 = 1533460707...7676737840$



$W = (w_j, t_j) =$

+34687399815	2
+34623434477	9
+34759421743	10

# Private Intersection-Sum (PIS)

🔑  $k_1 = 3435215506...4956620898$

Alice



V =

+34633946745
+34759421743
+34623434477
+34602305532

Información común: función hash  $H$  y  $p$

$H = \text{SHA512}$

$p = 5573842157...6869429667$  (3072 bits)

Bob

🔑  $k_2 = 1533460707...7676737840$

🔑  $pk = (193...939, 193...940)$

🔑  $sk = (112...709, 172...471)$



$W = (w_j, t_j) =$

+34687399815	2
+34623434477	9
+34759421743	10



# Private Intersection-Sum (PIS)

🔑  $k_1 = 3435215506...4956620898$

Alice



$V =$

+34633946745
+34759421743
+34623434477
+34602305532

Información común: función hash  $H$  y  $p$

$H = \text{SHA512}$

$p = 5573842157...6869429667$  (3072 bits)

Bob

🔑  $k_2 = 1533460707...7676737840$

🔑  $pk = (193...939, 193...940)$

🔑  $sk = (112...709, 172...471)$



$W = (w_j, t_j) =$

+34687399815	2
+34623434477	9
+34759421743	10

🔑  $pk$



# Private Intersection-Sum (PIS)

🔑  $k_1 = 3435215506...4956620898$

Alice



Información común: función hash  $H$  y  $p$

$H = \text{SHA512}$

$p = 5573842157...6869429667$  (3072 bits)

Bob

🔑  $k_2 = 1533460707...7676737840$

🔑  $pk = (193...939, 193...940)$

🔑  $sk = (112...709, 172...471)$

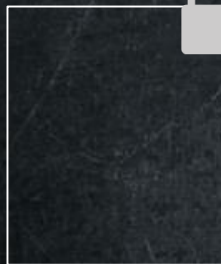


$V =$

+34633946745
+34759421743
+34623434477
+34602305532



$A = H(V)^{k_1} \bmod p =$

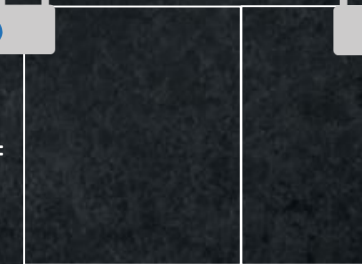


$W = (w_j, t_j) =$

+34687399815	2
+34623434477	9
+34759421743	10



$B = (H(w_j)^{k_2} \bmod p, E(t_j, pk)) =$



🔑  $pk$





# Private Intersection-Sum (PIS)

🔑  $k_1 = 3435215506...4956620898$

Alice



Información común: función hash  $H$  y  $p$

$H = \text{SHA512}$

$p = 5573842157...6869429667$  (3072 bits)

Bob

🔑  $k_2 = 1533460707...7676737840$

🔑  $pk = (193...939, 193...940)$

🔑  $sk = (112...709, 172...471)$



$V =$

+34633946745
+34759421743
+34623434477
+34602305532

$A = H(V)^{k_1} \bmod p =$

26515...6911
36026...74506
20035...62805
14643...79680

$W = (w_j, t_j) =$

+34687399815	2
+34623434477	9
+34759421743	10

$B = (H(w_j)^{k_2} \bmod p, E(t_j, pk)) =$

12501...46561
29114...17208
27704...63860

🔑  $pk$



# Private Intersection-Sum (PIS)

🔑  $k_1 = 3435215506...4956620898$

Alice



Información común: función hash  $H$  y  $p$

$H = \text{SHA512}$

$p = 5573842157...6869429667$  (3072 bits)

Bob

🔑  $k_2 = 1533460707...7676737840$

🔑  $pk = (193...939, 193...940)$

🔑  $sk = (112...709, 172...471)$



$V =$

+34633946745
+34759421743
+34623434477
+34602305532

$A = H(V)^{k_1} \bmod p =$

26515...6911
36026...74506
20035...62805
14643...79680

$W = (w_j, t_j) =$

+34687399815	2
+34623434477	9
+34759421743	10

$B = (H(w_j)^{k_2} \bmod p, E(t_j, pk)) =$

12501...46561	34...33
29114...17208	40...21
27704...63860	13...08

🔑  $pk$





# Private Intersection-Sum (PIS)

🔑  $k_1 = 3435215506...4956620898$

Alice



Información común: función hash  $H$  y  $p$

$H = \text{SHA512}$

$p = 5573842157...6869429667$  (3072 bits)

Bob

🔑  $k_2 = 1533460707...7676737840$

🔑  $pk = (193...939, 193...940)$

🔑  $sk = (112...709, 172...471)$



$V =$

+34633946745
+34759421743
+34623434477
+34602305532

$A = H(V)^{k_1} \bmod p =$

26515...6911
36026...74506
20035...62805
14643...79680

$W = (w_j, t_j) =$

+34687399815	2
+34623434477	9
+34759421743	10

$B = (H(w_j)^{k_2} \bmod p, E(t_j, pk)) =$

12501...46561	34...33
29114...17208	40...21
27704...63860	13...08

🔑  $pk$

$A$

# Private Intersection-Sum (PIS)

🔑  $k_1 = 3435215506...4956620898$

Alice



Información común: función hash  $H$  y  $p$

$H = \text{SHA512}$

$p = 5573842157...6869429667$  (3072 bits)

Bob

🔑  $k_2 = 1533460707...7676737840$

🔑  $pk = (193...939, 193...940)$

🔑  $sk = (112...709, 172...471)$



$V =$

+34633946745
+34759421743
+34623434477
+34602305532

$A = H(V)^{k_1} \bmod p =$

26515...6911
36026...74506
20035...62805
14643...79680

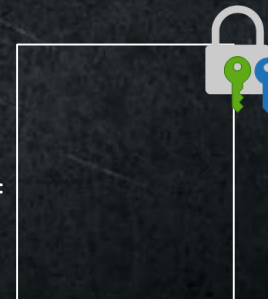
$W = (w_j, t_j) =$

+34687399815	2
+34623434477	9
+34759421743	10

$B = (H(w_j)^{k_2} \bmod p, E(t_j, pk)) =$

12501...46561	34...33
29114...17208	40...21
27704...63860	13...08

$Z = A^{k_2} \bmod p =$





# Private Intersection-Sum (PIS)

🔑  $k_1 = 3435215506...4956620898$

Alice



Información común: función hash  $H$  y  $p$

$H = \text{SHA512}$

$p = 5573842157...6869429667$  (3072 bits)

Bob

🔑  $k_2 = 1533460707...7676737840$

🔑  $pk = (193...939, 193...940)$

🔑  $sk = (112...709, 172...471)$



$V =$

+34633946745
+34759421743
+34623434477
+34602305532

$A = H(V)^{k_1} \bmod p =$

26515...6911
36026...74506
20035...62805
14643...79680

$W = (w_j, t_j) =$

+34687399815	2
+34623434477	9
+34759421743	10

$B = (H(w_j)^{k_2} \bmod p, E(t_j, pk)) =$

12501...46561	34...33
29114...17208	40...21
27704...63860	13...08

$Z = A^{k_2} \bmod p =$

13232...72966
32199...88159
25626...73418
55232...25729

🔑  $pk$

$A$

# Private Intersection-Sum (PIS)

  $k_1 = 3435215506...4956620898$

Alice



Información común: función hash  $H$  y  $p$

$H = \text{SHA512}$

$p = 5573842157...6869429667$  (3072 bits)

Bob

  $k_2 = 1533460707...7676737840$

  $pk = (193...939, 193...940)$

  $sk = (112...709, 172...471)$



$V =$

+34633946745
+34759421743
+34623434477
+34602305532

$A = H(V)^{k_1} \bmod p =$

26515...6911
36026...74506
20035...62805
14643...79680

$W = (w_j, t_j) =$

+34687399815	2
+34623434477	9
+34759421743	10

$B = (H(w_j)^{k_2} \bmod p, E(t_j, pk)) =$

12501...46561	34...33
29114...17208	40...21
27704...63860	13...08

$A$

$Z, B$

$Z = A^{k_2} \bmod p =$

13232...72966
32199...88159
25626...73418
55232...25729



# Private Intersection-Sum (PIS)

🔑  $k_1 = 3435215506...4956620898$

Alice



Información común: función hash  $H$  y  $p$

$H = \text{SHA512}$

$p = 5573842157...6869429667$  (3072 bits)

Bob

🔑  $k_2 = 1533460707...7676737840$

🔑  $pk = (193...939, 193...940)$

🔑  $sk = (112...709, 172...471)$



$V =$

+34633946745
+34759421743
+34623434477
+34602305532

$A = H(V)^{k_1} \bmod p =$

26515...6911
36026...74506
20035...62805
14643...79680

$W = (w_j, t_j) =$

+34687399815	2
+34623434477	9
+34759421743	10

$B = (H(w_j)^{k_2} \bmod p, E(t_j, pk)) =$

12501...46561	34...33
29114...17208	40...21
27704...63860	13...08

🔑  $pk$

$A$

$Z, B$

$enc = H(Z[t])^{k_1} =$

34...33
40...21
13...08

$Z = A^{k_2} \bmod p =$

13232...72966
32199...88159
25626...73418
55232...25729

# Private Intersection-Sum (PIS)

  $k_1 = 3435215506...4956620898$

Alice




Información común: función hash  $H$  y  $p$

$H = \text{SHA512}$

$p = 5573842157...6869429667$  (3072 bits)

Bob

  $k_2 = 1533460707...7676737840$

  $pk = (193...939, 193...940)$

  $sk = (112...709, 172...471)$



$V =$

+34633946745
+34759421743
+34623434477
+34602305532

$W = (w_j, t_j) =$

+34687399815	2
+34623434477	9
+34759421743	10

  $pk$

$A = H(V)^{k_1} \bmod p =$

26515...6911
36026...74506
20035...62805
14643...79680

$B = (H(w_j)^{k_2} \bmod p, E(t_j, pk)) =$

12501...46561	34...33
29114...17208	40...21
27704...63860	13...08

$A$

$Z, B$

$enc = H(Z[t])^{k_1} =$

48936...54612	34...33
25626...73418	40...21
32199...88159	13...08

$Z = A^{k_2} \bmod p =$

13232...72966
32199...88159
25626...73418
55232...25729



# Private Intersection-Sum (PIS)

🔑  $k_1 = 3435215506...4956620898$

Alice



Información común: función hash  $H$  y  $p$

$H = \text{SHA512}$

$p = 5573842157...6869429667$  (3072 bits)

Bob

🔑  $k_2 = 1533460707...7676737840$

🔑  $pk = (193...939, 193...940)$

🔑  $sk = (112...709, 172...471)$



$V =$

+34633946745
+34759421743
+34623434477
+34602305532

$A = H(V)^{k_1} \bmod p =$

26515...6911
36026...74506
20035...62805
14643...79680

$W = (w_j, t_j) =$

+34687399815	2
+34623434477	9
+34759421743	10

🔑  $pk$

$B = (H(w_j)^{k_2} \bmod p, E(t_j, pk)) =$

12501...46561	34...33
29114...17208	40...21
27704...63860	13...08

$A$

$Z, B$

$enc = H(Z[t])^{k_1} =$

48936...54612	34...33
25626...73418	40...21
32199...88159	13...08

$Z = A^{k_2} \bmod p =$

13232...72966
32199...88159
25626...73418
55232...25729



# Private Intersection-Sum (PIS)

🔑  $k_1 = 3435215506...4956620898$

Alice



Información común: función hash  $H$  y  $p$

$H = \text{SHA512}$

$p = 5573842157...6869429667$  (3072 bits)

Bob

🔑  $k_2 = 1533460707...7676737840$

🔑  $pk = (193...939, 193...940)$

🔑  $sk = (112...709, 172...471)$



$V =$

+34633946745
+34759421743
+34623434477
+34602305532

$A = H(V)^{k_1} \bmod p =$

26515...6911
36026...74506
20035...62805
14643...79680

$enc = H(Z[t])^{k_1} =$

48936...54612	34...33
25626...73418	40...21
32199...88159	13...08
	35...27

$c = \text{suma}(\text{■}) =$

$W = (w_j, t_j) =$

+34687399815	2
+34623434477	9
+34759421743	10

$B = (H(w_j)^{k_2} \bmod p, E(t_j, pk)) =$

12501...46561	34...33
29114...17208	40...21
27704...63860	13...08

$Z = A^{k_2} \bmod p =$

13232...72966
32199...88159
25626...73418
55232...25729

🔑  $pk$

$A$

$Z, B$



# Private Intersection-Sum (PIS)

🔑  $k_1 = 3435215506...4956620898$

Alice



Información común: función hash  $H$  y  $p$

$H = \text{SHA512}$

$p = 5573842157...6869429667$  (3072 bits)

Bob

🔑  $k_2 = 1533460707...7676737840$

🔑  $pk = (193...939, 193...940)$

🔑  $sk = (112...709, 172...471)$



$V =$

+34633946745
+34759421743
+34623434477
+34602305532

$W = (w_j, t_j) =$

+34687399815	2
+34623434477	9
+34759421743	10

🔑  $pk$

$A = H(V)^{k_1} \bmod p =$

26515...6911
36026...74506
20035...62805
14643...79680

$B = (H(w_j)^{k_2} \bmod p, E(t_j, pk)) =$

12501...46561	34...33
29114...17208	40...21
27704...63860	13...08

$A$

$Z, B$

$enc = H(Z[t])^{k_1} =$

48936...54612	34...33
25626...73418	40...21
32199...88159	13...08

$Z = A^{k_2} \bmod p =$

13232...72966
32199...88159
25626...73418
55232...25729

$c = \text{suma}(\blacksquare) =$

	35...27
--	---------

$C$



# Private Intersection-Sum (PIS)

  $k_1 = 3435215506...4956620898$

Alice



Información común: función hash  $H$  y  $p$

$H = \text{SHA512}$

$p = 5573842157...6869429667$  (3072 bits)

Bob

  $k_2 = 1533460707...7676737840$

  $pk = (193...939, 193...940)$

  $sk = (112...709, 172...471)$



$V =$

+34633946745
+34759421743
+34623434477
+34602305532

$W = (w_j, t_j) =$

+34687399815	2
+34623434477	9
+34759421743	10

  $pk$

$A = H(V)^{k_1} \bmod p =$

26515...6911
36026...74506
20035...62805
14643...79680

$B = (H(w_j)^{k_2} \bmod p, E(t_j, pk)) =$

12501...46561	34...33
29114...17208	40...21
27704...63860	13...08

$A$

$Z, B$

$enc = H(Z[t])^{k_1} =$

48936...54612	34...33
25626...73418	40...21
32199...88159	13...08

$Z = A^{k_2} \bmod p =$

13232...72966
32199...88159
25626...73418
55232...25729

$c = \text{suma}(\text{■}) =$

	35...27
--	---------

$C$

 19



# Conclusiones

- Tanto la criptografía homomórfica como la computación multiparte ofrecen un **sinfín de aplicaciones**, trabajando sobre el dominio cifrado “garantizando” **privacidad y seguridad**.
- Estas áreas son de gran relevancia en investigación, proponiendo cada vez **protocolos más rápidos y eficientes**, e incluso, dando lugar a **librerías fáciles de usar** y con las que implementar nuestras propias aplicaciones.
- Hay que ser consciente de las **ventajas y desventajas** que ofrecen estas tecnologías para implementar un caso, atendiendo al tipo de adversario que nos enfrentamos, requisitos de seguridad y velocidad.



# Referencias (I)

[Acar2018] Acar, A., Aksu, H., Uluagac, A. S., & Conti, M. (2018). A Survey on Homomorphic Encryption Schemes. ACM Computing Surveys, 51(4), 1–35. <https://doi.org/10.1145/3214303>

[Albrecht2015] Albrecht, M. R., Player, R., & Scott S. (2015). On the concrete hardness of Learning with Errors. <https://eprint.iacr.org/2015/046.pdf>

[Albrecht2018] Albrecht, M., Chase, M., Chen, H., Ding, J., Goldwasser, S., Gorbunov, S., Vaikuntanathan, V. (2018) Homomorphic Encryption Standard. <https://eprint.iacr.org/2019/939.pdf>

[Archer2017] Archer, D., Chen, L., Cheon, J. H., Gilad-Bachrach, R., Hallman, R. A., Huang, Z., Wang, S. Applications of homomorphic encryption. [http://homomorphicencryption.org/white\\_papers/applications\\_homomorphic\\_encryption\\_white\\_paper.pdf](http://homomorphicencryption.org/white_papers/applications_homomorphic_encryption_white_paper.pdf)

[Bloom1970] Bloom, B. H., & H., B. (1970). Space/time trade-offs in hash coding with allowable errors. Communications of the ACM, 13(7), 422–426. <https://doi.org/10.1145/362686.362692>

[Cheon2017] Cheon, J. H., Kim, A., Kim, M., & Song, Y. Homomorphic Encryption for Arithmetic of Approximate Numbers. <https://eprint.iacr.org/2016/421.pdf>



# Referencias (II)

[DeCristofaro2010] De Cristofaro, E., & Tsudik, G. (2010). *Practical Private Set Intersection Protocols with Linear Complexity*. [https://doi.org/10.1007/978-3-642-14577-3\\_13](https://doi.org/10.1007/978-3-642-14577-3_13)

[Evans2018] Evans D., Kolesnikov V. & Rosulek M., A Pragmatic Introduction to Secure Multi-Party Computation. NOW Publishers, 2018. <https://securecomputation.org/docs/pragmaticmpc.pdf>

[Holden2008] Holden, J. The Pohlig-Hellman exponentiation cipher as a bridge between classical and modern cryptography. <https://www.rose-hulman.edu/~holden/Preprints/pohlig-hellman.pdf>

[Ion2019] Ion, M., Kreuter, B., Nergiz, A. E., Patel, S., Raykova, M., Saxena, S., Yung, M. On Deploying Secure Computing Commercially: Private Intersection-Sum Protocols and their Business Applications. <https://eprint.iacr.org/2019/723.pdf>

[Kales2019] Kales, D., Rechberger C., & Schneider, T. (2019). Mobile Private Contact Discovery at Scale. <https://eprint.iacr.org/2019/517.pdf>

[Kissner2005] Kissner, L., & Song, D. (2005). Privacy-Preserving Set Operations. <https://www.cs.cmu.edu/~leak/papers/set-tech-full.pdf>



# Referencias (III)

[Miyaji2015] Miyaji, A., & Nishida, S. (2015). A Scalable Multiparty Private Set Intersection.  
[https://doi.org/10.1007/978-3-319-25645-0\\_26](https://doi.org/10.1007/978-3-319-25645-0_26)

[Peng2019] Peng, Z. (2019). Danger of using fully homomorphic encryption: A look at Microsoft SEAL.  
<https://arxiv.org/pdf/1906.07127.pdf>

[Pohlig1978] Pohlig, S., & Hellman, M. (1978). An improved algorithm for computing logarithms over  $GF(p)$  and its cryptographic significance. IEEE Transactions on Information Theory, 24(1), 106–110.  
<https://doi.org/10.1109/TIT.1978.1055817>

[Sri2018] Sri, S., Vepakomma, P., Raskar, R., Ramachandra, R., Bhattacharya, S., Ai, S. (2018). A Review of Homomorphic Encryption Libraries for Secure Computation. <https://arxiv.org/pdf/1812.02428.pdf>



# Sharing Privacy: From 0 to PSI (Private Set Intersection)

Dr. Alfonso Muñoz (@mindcrypt) - [alfonso.munoz2.next@bbva.com](mailto:alfonso.munoz2.next@bbva.com)

José Ignacio Escribano - [joseignacio.escribano.pablos.next@bbva.com](mailto:joseignacio.escribano.pablos.next@bbva.com)

RootedCon Valencia 2019

¡Muchas gracias por vuestra atención!