Function Preserving Encryption Overview

Stan Trepetin
(OptimalCipher)

Data Security/Analytics Problem

The impact of security breaches continues to increase:

- Number of records compromised due to breaches rose from 169.1 million in 2015 to *15.1 billion* in 2019.
- The global average total cost of a breach has risen from $3.50 million in 2014 to $3.92 million in 2019.

Sources: Statista: Annual number of data breaches and exposed records in the United States from 2005 to 2019; The Definitive Cyber Security Statistics Guide for 2020; DataInsider: What's the Cost of a Data Breach in 2019?

Data Security/Analytics Problem

Reason for many security breaches:

- Data is typically not collected for its own sake--organizations need to *analyze* their collected information.
- But to support analytics--data must be in a decrypted state.
- But in plaintext form, data can be stolen by hackers or viruses (theft of files, memory attacks, etc).

To support today's increasing data-driven processes, a platform is needed which strongly, continually protects information, while permitting data analysis.

Function Preserving Encryption (FPE) Value Proposition

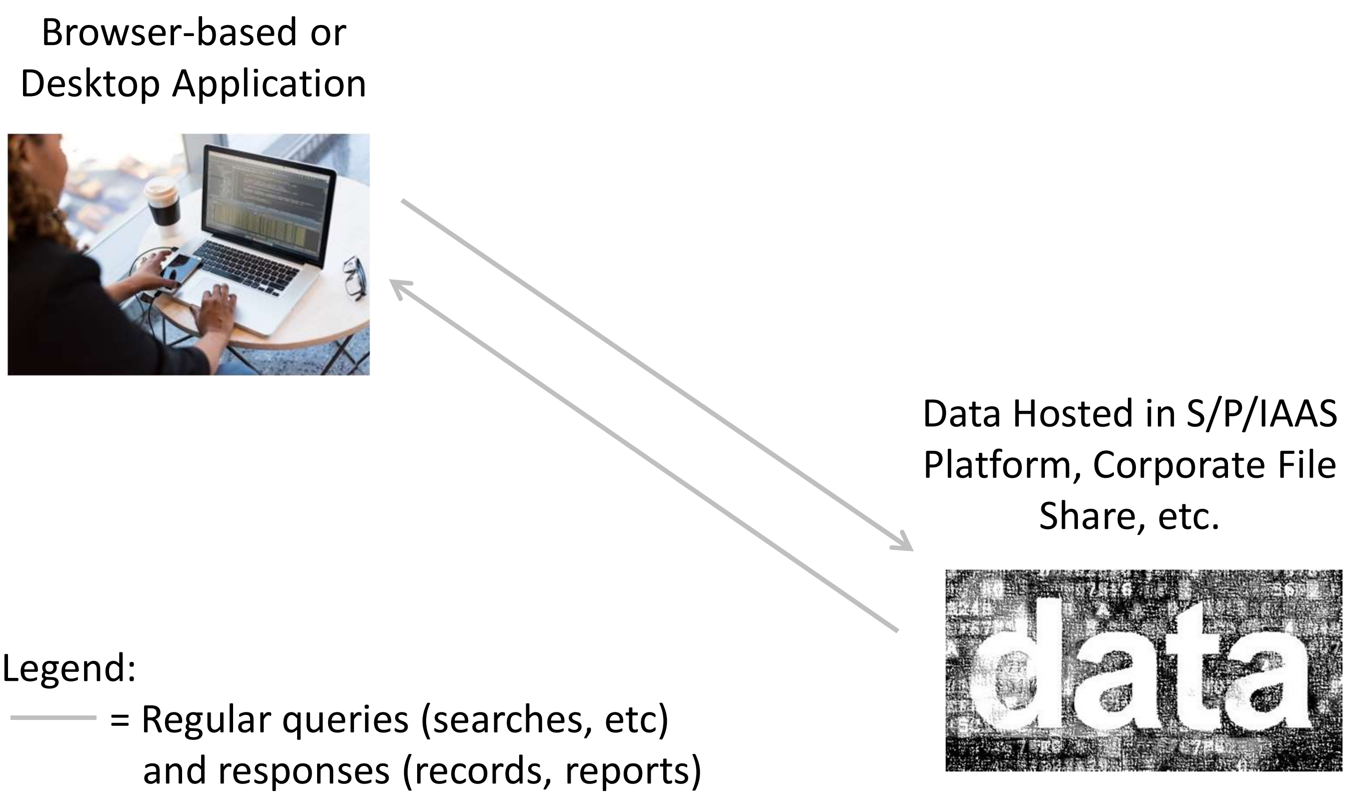
FPE is comprised of standard and new encryption and tokenization techniques which provide comprehensive analysis of data while the data are entirely encrypted/tokenized. FPE:

- Fully encrypts data, yet allows applications to search, sort, and perform mathematics and statistics on the encrypted data.
- Protects data in numerous hosting environments, e.g. in cloud, in data center, on mobile devices, etc.
- Fully encrypts all data rather than just some data--thereby preventing possibilities of 'data re-identification'. This also eases compliance with regulations, which prescribe different controls for different data categories.
- May provide a lower Total Cost of Ownership for enterprise encryption; as a single administrative console, instead of different point encryption solutions, is used to control multiple encrypted domains.
- Protects the data--not the systems the data lives on. No matter where the data travels to, it remains secure. Only authorized users with the appropriate decryption key can decrypt it.

Some Current FPE Examples

- Cloud Access Security Broker platforms (CASBs support FPE for popular SAAS platforms. However, they do not handle encrypted mathematics).
- Tokenization platforms (permit searching for equality among tokenized data).
- Microsoft SEAL Homomorphic Encryption (open source library supports addition and multiplication over encrypted numbers--but search over encrypted data is difficult to do).
- Order-Preserving Encryption (open-source systems that permit searching for equality as well as sorting over encrypted data (e.g., MIN, MAX functionality)--but do not support encrypted mathematics).

Typical Data Request


Browser-based or Desktop Application
Data Hosted in S/P/IAAS Platform, Corporate File Share, etc.
data

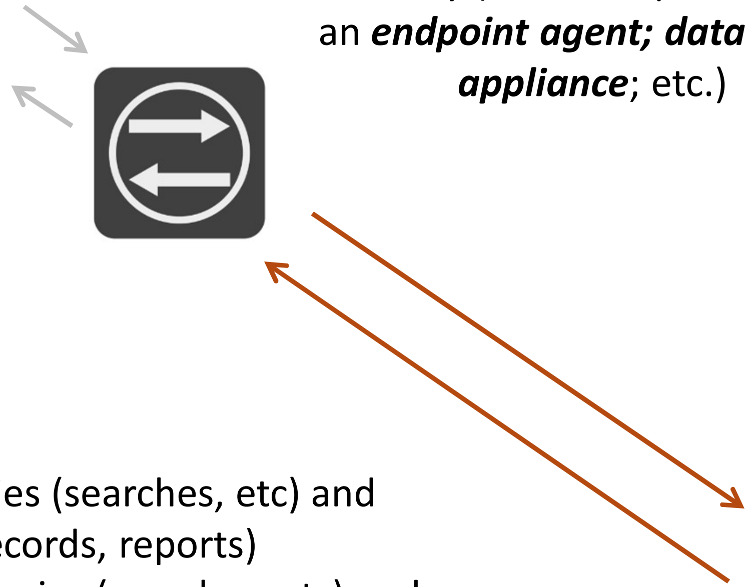
Legend:

—— = Regular queries (searches, etc) and responses (records, reports)

FPE-Encrypted Data Request

Browser-based or Desktop Application

FPE Proxy (can be implemented as an ***endpoint agent; data center appliance***; etc.)



Encrypted Data Hosted in S/P/IAAS Platform, Corporate File Share, etc.

Legend:

— = Regular queries (searches, etc) and responses (records, reports)

— = Encrypted queries (searches, etc) and responses (records, reports)

FPE Implementation Details

Operations:

- Step 1: Data is encrypted and uploaded to host environment.
- Step 2: Regular application/client queries are intercepted and transformed into "encrypted queries" (with potentially different structure, and encrypting any arguments), which are forwarded to the hosting environment.
- Step 3: The hosting environment evaluates request, sending encrypted result back to application/client.
- Step 4: The application/client intercepts and decrypts result, displaying the plaintext result to the user.

Example:

- Step 1: Plaintext table encrypted and sent to database host.

Name	Blood Tests
John	4
Sally	7

Name column Tokenized. *Blood Tests* column transformed via Benaloh encryption. (See Appendix for more detailed description of Benaloh, here just represented as E(a) when encrypting number a).

Name	Blood Tests
8H{"!	E(4)
V$j–0	E(7)

- Step 2:
  - Suppose the application needs a SUM of the Blood Tests column:

SELECT SUM(Blood Tests) FROM <table>

Gets intercepted by application/client and converted to SQL on right….. before being sent to database host

SELECT **EXP(SUM(LOG(Blood Tests)))** FROM <table>

(This conversion happens because Benaloh permits adding numbers by multiplying their encrypted values. However, because in SQL, multiplying data within the same column is challenging to do, we can't easily multiply E(4) by E(7) (e.g. as in this example). We have to use equivalent functions--Exponentiation of the Sum of the Logarithms. Benaloh's encryption and the use of these 'equivalent' functions are further described in the Appendix).

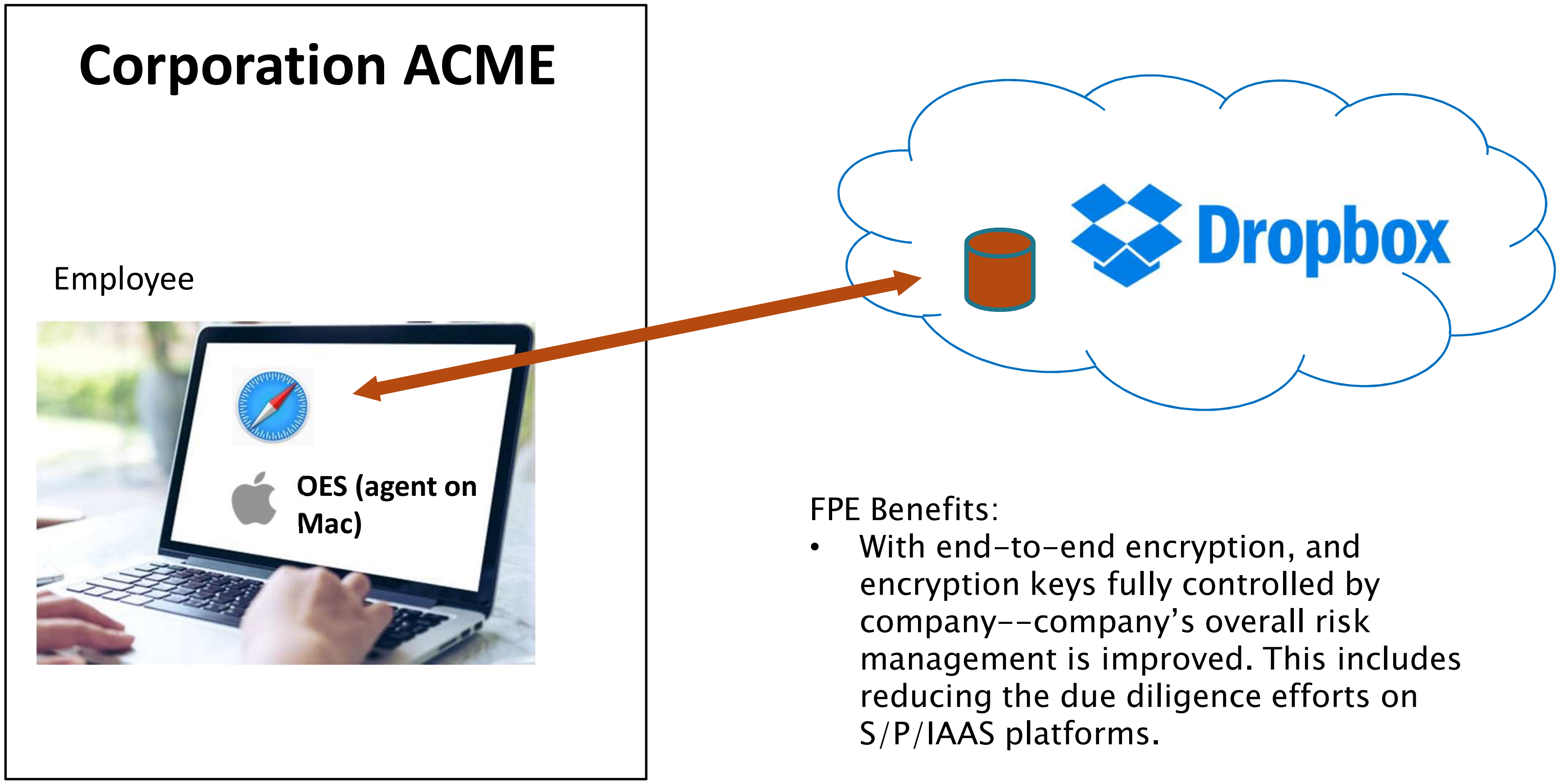
- Steps 3 and 4:
  - When requesting SUM of Blood Tests column:

The encrypted result computed and returned, EXP(LOG(E(4)) + LOG(E(7))), as per Step 2

Gets intercepted by application/client and decrypted to result on right….. before being displayed to user. (D on the right is the standard Benaloh decryption function. See additional information on Benaloh in the Appendix).

D(E(11)) = 11

Use Case 1: End-To-End Encryption

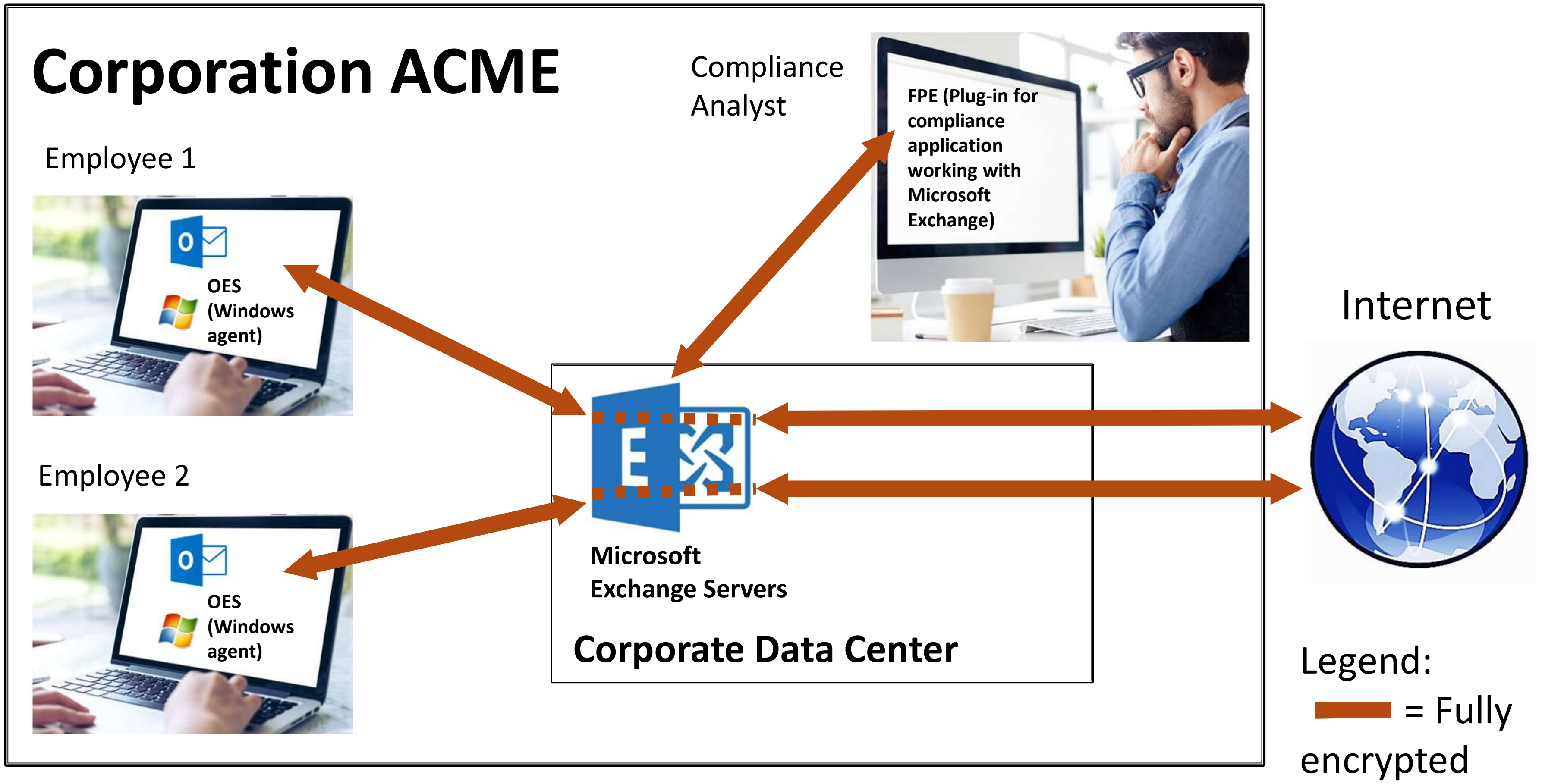


FPE Benefits:

- With end-to-end encryption, and encryption keys fully controlled by company--company's overall risk management is improved. This includes reducing the due diligence efforts on S/P/IAAS platforms.

Legend: 🟥 = Fully encrypted data/traffic

Use Case 2: Encrypted Data Analysis



Legend:
▬ = Fully encrypted data/traffic

FPE Benefits:

- Analysis of encrypted emails for compliance purposes before emails leave company (e.g. privacy-protective policy enforcement).

Use Case 3: Obviating Data Classification

Typical Data Classification

Employee Last Name	Employee First Name	Employee Age	Employee Email
Smith	Bob	55	bob@acme.com
Kline	Susan	43	susan@acme.com
Jones	Philip	63	philip@acme.com

Does GDPR require protecting Employee Email?
Does HIPAA require protecting Employee Age?
Is my company's policy clear on protecting Employee Last Name?
Etc.

Data Classification Under FPE

Employee Last Name	Employee First Name	Employee Age	Employee Email
8*(@31	Hj+=;]	11	KS2%_+
\}~d$#	Nx^^%@	88	64G:?2Z
pdW,{+	D05=!z	24	%0^`|=8f<

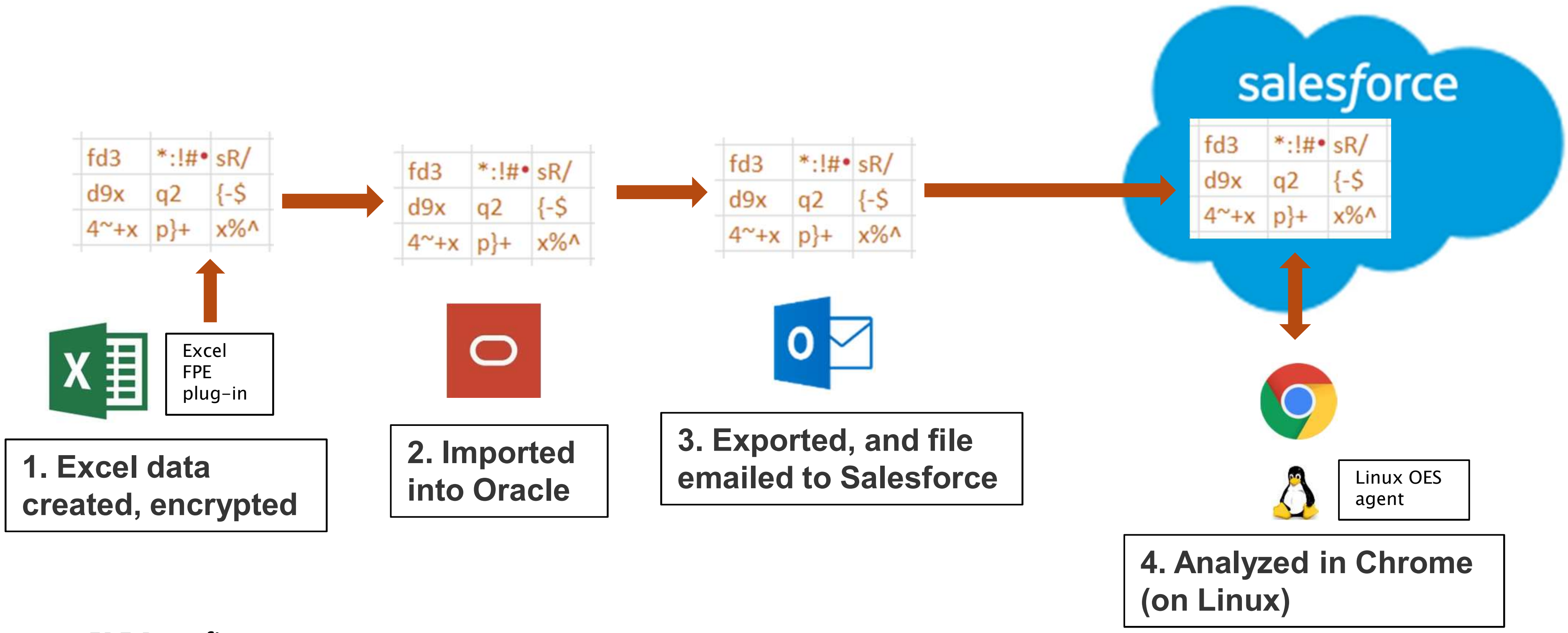
No substantial need for Data Classification, since within a system, virtually all data is encrypted. In essence--no "sensitive data" per se remains.

Legend:
🟥 = Fully encrypted data

FPE Benefits:

- Substantially less time required for Data Classification activities--since FPE considers all data to be at the highest sensitivity level, and applies one of the best security controls to data: encryption.

Use Case 4: Simplification of the Encryption Ecosystem


salesforce
fd3 *:!#• sR/
d9x q2 {-$
4~+x p}+ x%^
Excel FPE plug-in
1. Excel data created, encrypted
2. Imported into Oracle
3. Exported, and file emailed to Salesforce
Linux OES agent
4. Analyzed in Chrome (on Linux)

FPE Benefits:

- Encryption ecosystem Total Cost of Ownership can be reduced since only one encryption system--FPE--is utilized (instead of using Excel's **password-protected encryption**; Oracle's **Transparent Data Encryption (TDE)**; etc).
- Relying on a single system also obviates the need to decrypt and re-encrypt data moving between platforms (e.g., removing Excel's password protection to import data into and TDE re-encrypt it within Oracle). This reduces security breach risk as data is never in a decrypted state; and also speeds data throughput.

Legend:
▬ = Fully encrypted data/traffic

Appendix: Benaloh Encryption and SQL

Benaloh Encryption

- Under parameters modulus=n, base=g, blocksize=c, and r=some random number within [0,n-1], the Benaloh encryption of number a is E(a) = (g^a)*(r^c) mod n.
- Such a scheme preserves homomorphic addition by multiplying the encrypted values:
  1. To calculate a+b, we multiply E(a) by E(b):

     E(a)*E(b) =

     (g^a)*(r1^c) mod n * (g^b)*(r2^c) mod n =

     g^(a+b)*(r1*r2)^c mod n.
  2. Since the last expression is just E(a+b)--by the definition of Benaloh encryption--by decrypting this value we get the plaintext sum a+b.
- More information on Benaloh encryption and its properties is available here: https://en.wikipedia.org/wiki/Benaloh_cryptosystem, https://en.wikipedia.org/wiki/Homomorphic_encryption.

Implementing Benaloh Encryption in SQL

- If we've implemented Benaloh in SQL/RDBMS and want to add numbers homomorphically, we need to multiply the encrypted values. Yet if values are in the same column, we'll have to perform the multiplication indirectly, since SQL typically does not permit multiplying values by each other within the same column.
- We can employ the Logarithm property: x = e^(ln(x)). Under this rule, we have

  x*y = e^(ln(x*y)) = e^(ln(x)+ln(y)) (by the properties of Logarithmic addition)
- Therefore, to add a and b, we can take the log of E(a) and log of E(b), add these values, and raise e to this sum. In SQL, this calculation can be readily set up as:

  ```
  SELECT EXP(SUM(LOG(<column name>))) from <table name> WHERE <selection criteria picking out just E(a) and E(b)>
  ```
- When decrypting the result of this statement, we'll obtain our plaintext a+b.
- The above process can be used to homomorphically add any subset of values in a column encrypted by Benaloh.

Questions?

Stan Trepetin

info@optimalcipher.com