Microsoft

Accurate data classification using Exact Data Matching Part 2

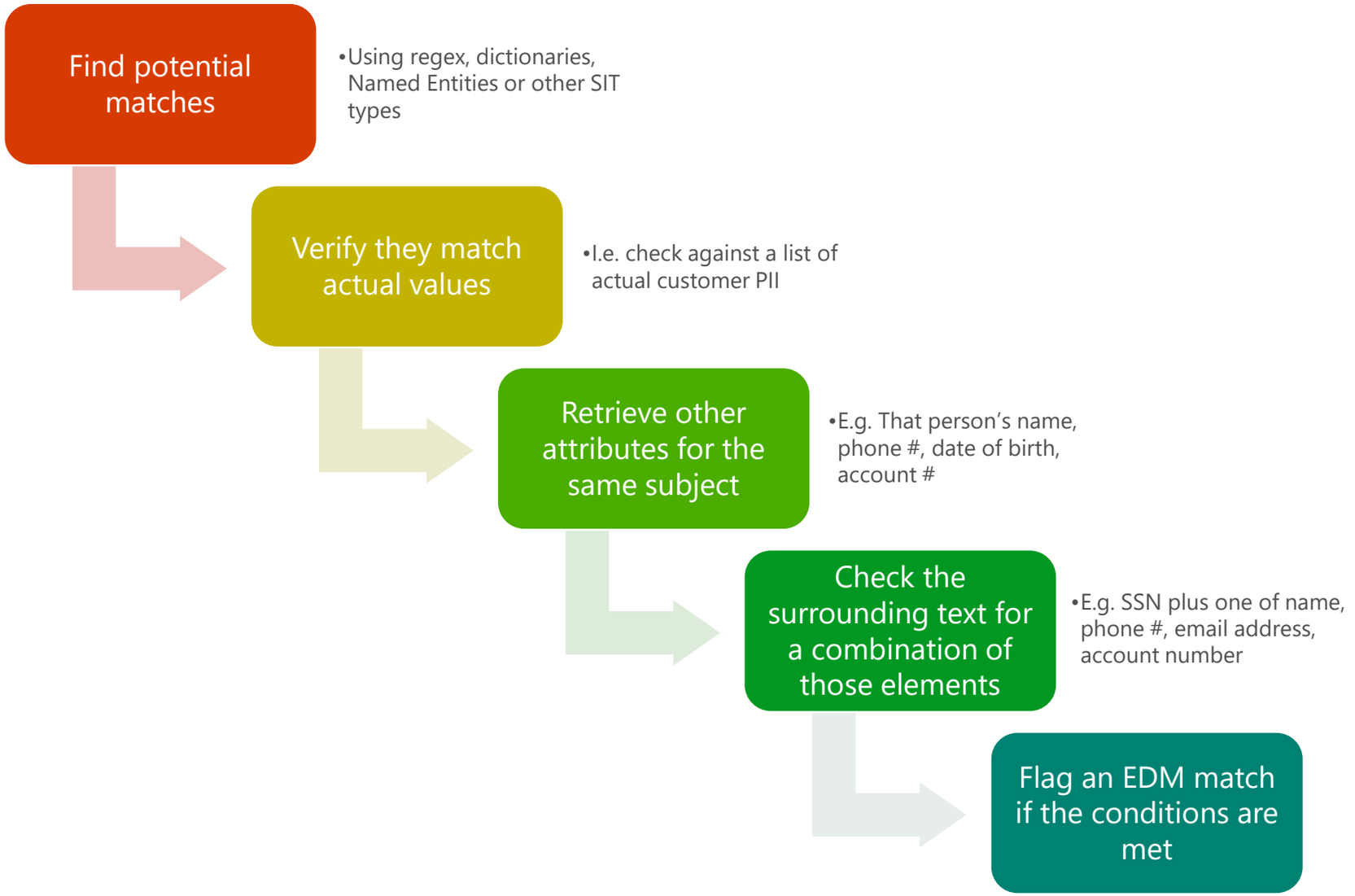
Enrique Saggese
Principal Program Manager, CxE

Microsoft Purview
Information Protection and Compliance
Microsoft Corp.

Agenda

- What is EDM
- EDM under the hood
- How to configure EDM
- EDM in specific industries
- Tips, tricks and advanced techniques for deploying EDM
- Sneak peek: the new EDM wizard

EDM at work

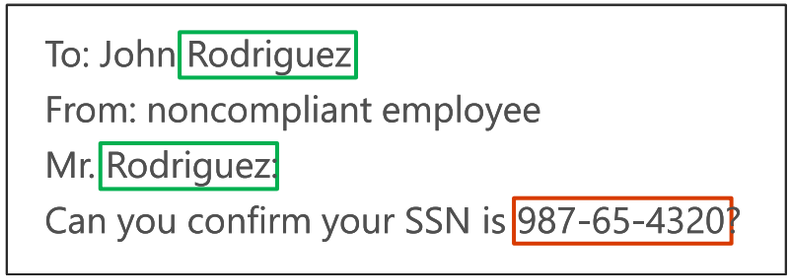

Find potential matches
•Using regex, dictionaries, Named Entities or other SIT types
Verify they match actual values
•I.e. check against a list of actual customer PII
Retrieve other attributes for the same subject
•E.g. That person's name, phone #, date of birth, account #
Check the surrounding text for a combination of those elements
•E.g. SSN plus one of name, phone #, email address, account number
Flag an EDM match if the conditions are met

How hashing is used in EDM

SSN	LastName	DoB	Account#
987-65-4320	Rodriguez	2/14/1980	ABC19838372
078-05-1120	Smith	3/22/1957	BAD38229209
219-09-9999	Zhang	11/10/2001	AAB39383894

Hash and Upload

SSN	LastName	DoB	Account#
56f6e20696e2	6f6e2069634	0636f6d70617	2e4e45542c20
4b177e0db1d	82517c9053d	9e7280c96dd	39750ed0c4e
ebc4103dda7	6e206f662073	580e00857dd	3893eb087db



Find candidate

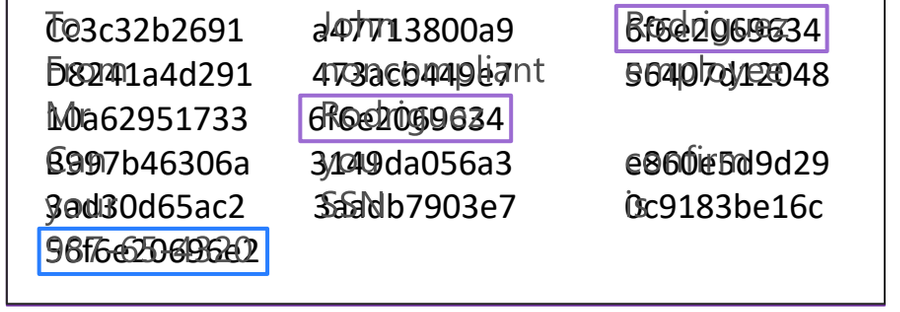
987654320

Hash

56f6e20696e2

Table lookup

Proximity search



Deploying EDM

EDM Best practices

General best practices

Creating your data table:

#1: Use TSV file for your sensitive file, not CSV

- If records have commas in values (e.g. street address) or single or double quotes (e.g. O'Connor, John "Hannibal" Smith, etc.) CSV is tricky to get right. TSV rarely causes problems even in these cases.
- Surround all columns within double quotes just for redundancy, EDM will strip them out before hashing.

#2: Sanitize your data *automatically*

- Identify issues with your table such as corrupt data, fields that need to be divided or surrounded in quotes, etc., and take note of the issues.
- Either convince the data owners to fix it up at the source, or build a script to fix the data in bulk, you don't want to be doing it manually on a weekly basis.

Hashing and importing your data table:

#3: Install the EDM upload agent in a custom folder

- If you install it in the default folder you will need admin privileges to run it, since it is under Program Files and the tool writes logs in the same folder.

#4: Validate your data before hashing to detect potential format issues (e.g. missing or extra columns)

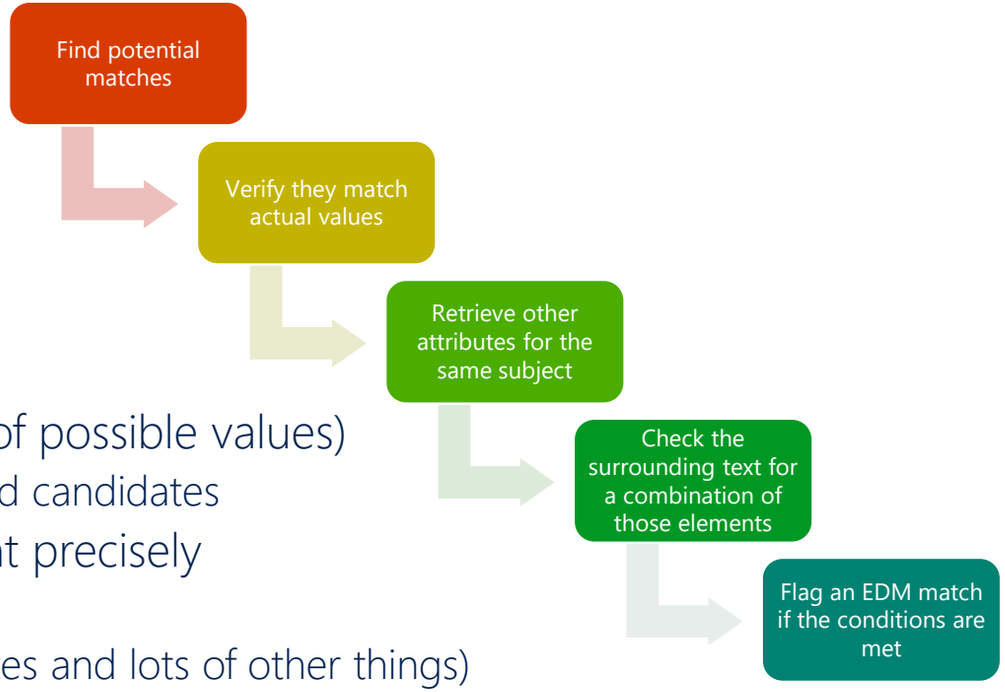
- `EdmUploadAgent.exe /ValidateData /DataFile [data file] /Schema [schema file]`

#5: Separate hash and upload processes

- The EDM table is extremely sensitive before hashing, do not put it in an internet-facing computer
- Hash in an internal machine, delete the original file, copy the hashed file to an internet facing computer and upload from there.
- Build a script that does it for you every time!

The challenge with the primary element

- Each SIT pattern has a primary evidence field
- Match candidates will be matched using a SIT first



- That field must meet *all* the following conditions:
  - Marked in the schema as searchable (so it's indexed)
  - The values must be "relatively" unique (e.g. have many thousands of possible values)
    - Date of birth, gender, marital status, first name, nationality, e.g. are all bad candidates
  - The values must be detectable using a SIT that matches the content precisely
    - Must detect all values
    - Not too many false positives (e.g. "four-digit number" will be fond in dates and lots of other things)
    - Not too common in content (e.g. no more than 100 documents/emails per second with matches on average)
  - The values in the table must match what's in content as-is in the documents (e.g. if Full name is listed as Firstname Lastname, it will not match "Lastname, Firstname").

Key considerations for the “trigger” SIT

- The SIT used for a primary element must not be too frequently present in content
  - E.g. not in every document or email generated.
  - Keep in mind that document metadata and text in email headers are also included in matching!
    - Every email and document has multiple dates, email addresses, GUIDs, IP addresses, names, etc.. Make sure you are not using a SIT that will detect those!
      - SIT might also be improperly firing because it is finding “substrings”, e.g. \d{6} will detect six consecutive digits within another string, like a GUID, \b\d{6}\b will only detect six digits alone.
- The SIT must match the whole string as present in the table and nothing else
  - Or else, it will produce a hash that’s different from what is stored.
  - E.g. [a-z]+\@[a-z]+\.[a-z]+ will detect only the highlighted part in the email address: john.smith@company.co.uk
  - A regex starting and/or ending with \s will include the space character as part of the match! Use \b instead.
- The SIT must match with and without any optional delimiters
  - The “ignored delimiters” option only strips the characters after a candidate is detected
    - E.g. if Ignored delimiters is set to “-”, the string 123-45-6789 will be stripped of the dash before hashing, but if the SIT is looking for \d{9} it won’t even flag it as a candidate.

Question: why do we need a SIT to match EDM

- Matching each individual string in a document against a table with tens of millions of rows is incredibly wasteful.*
  - It would waste >99.99% of the lookups in things that could never match. Humans would never do that, neither should computers.
  - If you are trying to find out if a document includes your SSN, do you skim through it until you find a long number first, or you compare each individual word against your SSN digits?
- Traditionally this problem was solved by throwing hardware at it. I.e. you pay for inefficiency in CPUs, energy, etc.
  - In the cloud, you would pay for it in higher service prices. .
- To avoid this, we ask you to tell us "how does the data we are looking for look like?"
  - A SIT is the perfect method for doing that!

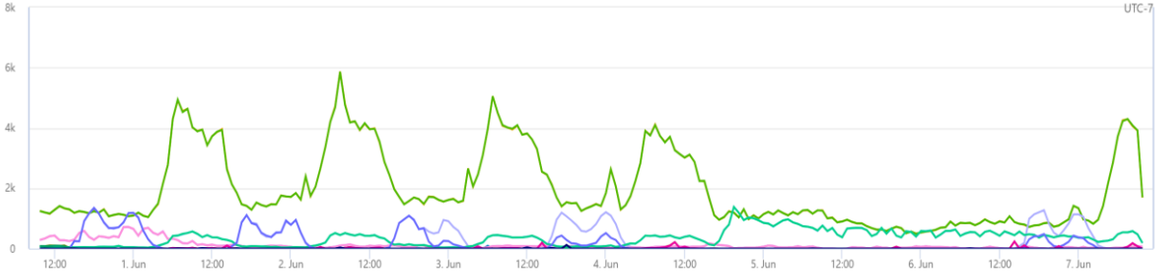
* For number nerds like me, the effort to compare all strings in a document with n characters against a table with m rows amounts to $[n^2+n]/2$ hashing computations and $[n^2+n]/2*m$ comparisons. Looking up the text in this PPT against an indexed table with 100 million rows would require 3 billion comparisons. And companies generate A LOT of content.

What happens if you choose the wrong SIT

- If the wrong SIT is used to trigger EDM matching:

1) It may cause missed detections because the system is too busy processing random strings that won't match.

- If a SIT causes an average of more than ~100 matches per second in your tenant, it is likely a bad candidate to use as trigger for a primary element (limit is enforced at ~15000 documents with matches per minute)
- Be aware of peaks in load, plan for the worst.
- If there's significant throttling, we may contact you.



2) If each value in the primary element column is repeated across too many rows, it may cause processing to timeout before all rows are analyzed.

- E.g. if you have a "birth year" column, there could be lots of dates in each document for matching, returning tens of thousands of matching rows each, all evidence columns for each need to be compared to the hashes of all words surrounding each of those dates.

3) An improperly defined SIT might fail to detect valid matches (e.g. an account number SIT might not accept numbers with dashes or in lower case), so EDM doesn't get to try them against the table.

4) An "unbound" regex might match a substring or superstring of what's in the table, so hashes won't match. E.g. a partial email address.

5) It might match something that is spelled out or ordered differently from what's in the table, so the hashes won't agree.

IF you get something from this webinar, let it be NOT TO USE a "wildcard" regex as a SIT trigger!!!! E.g.:

- \w+ (matches every string)
- \b\w+\b (matches every word)
- [A-Za-z0-9]+ (matches any string)
- \d+ (matches any number of any length)

NOTE: if an EDM SIT is throttled in your tenant, ALL EDM SITS ARE THROTTLED IN YOUR TENANT. So even an unused EDM SIT might cause EDM to not be functional. Don't leave broken test EDM SITs around.

Exercise: define the primary element

LastName	Firstname	SSN	DoB	Account#	Address
Rodriguez	John	987-65-4320	2/14/1980	ABC19838372	24827 NE 23rd St., Redmond, WA
Smith	Edward	078-05-1120	3/22/1957	BAD38229209	873 Newport Ct., Kent, WA
Zhang	Li	219-09-9999	11/10/2001	AAB39383894	9982 High Road Circle, Springfield, CO

- First question: which permutations are needed?
  - E.g. Last Name and (SSN or Account # or Address), SSN and DoB, Account # and (DoB, Address or SSN).
  - Depends on your business needs.
  - Let's say you need to detect those three combinations above.
- What's the ideal set of primary elements?
  - Must be identifiable via a SIT
    - Excludes first name, DoB and Address, at least without using advanced techniques to be discussed later.
  - Must be relatively unique
    - Excludes first name and DoB
  - Must not be present in too great numbers unless it is what I need to detect
    - Excludes dates, last names and first names
- Ideal options are SSN and Account#. All permutations except for "Last Name and Address" include one of these two.
  - We'll discuss what to do for that specific combination later.

Exercise: choosing the right SIT

LastName	Firstname	SSN	DoB	Account#	Address
Rodriguez	John	987-65-4320	2/14/1980	ABC198383	24827 NE 23rd St., Redmond, WA
Smith	Edward	078-05-1120	3/22/1957	BAD382292	873 Newport Ct., Kent, WA
Zhang	Li	219-09-9999	11/10/2001	AAB393838	9982 High Road Circle, Springfield, CO

- For SSN we *could* use the built-in SSN
  - But then it would require a word like SSN to be present in the document AND within 300 characters of the SSN value.
  - In large tables, that may not be true or only true for the first few rows.
  - Solution: clone and edit the SSN SIT and remove the requirement for additional evidence (keywords).
- For Account #, we will have to create a regex-based SIT.
  - So, \W+, right?
  - NO! That will cause too many matches to process and potentially saturate our ability to detect sensitive data. Use a custom regex, e.g.: [A-Za-z]{3}\d{6}
  - But that will also detect something like b3afd387344d (something that may be part of a GUID, for example). So **ALWAYS** delimit your regexes: **\b**[A-Za-z]{3}\d{6}**\b**

So I can only use columns that match a simple regex?

NO!

First off, you may not need to use it as a primary element, and use the tricky column as "additional evidence" with another column as primary.

If you *have* to use it as a primary element:

You can use a regular expression with additional conditions built-in:

E.g. an email address regex that excludes those in the TO and CC lines:

(?m)(?<!((From|To|CC): ([a-zA-Z0-9. <>@\.\-\(\);])*))([a-zA-Z0-9.-]+\@[a-zA-Z0-9.]+\.[a-zA-Z]{2,15})\b

You can use a "Loose" regular expression with keywords:

E.g. \b\d{6,9}\b, but with the keywords "account number", "account ID", or "acct#" nearby.

You can use multiple regexes (in different patterns or as a combined regex joined by an OR condition)

E.g. you want to detect account numbers in multiple different formats: ABC12345, A12345XY, 12345678901234 , A1B2C3D123456, and more

Bad option: single regex pattern that detects all of them (and more): [A-Z\d]{8,14}

Better option: combine regexes for each format via "|":

\b([A-Z]{3}\d{5})|([A-Z]\d{5}\[XY]{2}|\d{14}|([A-Z]\d){4}\d{5})\b

But more importantly: SIT <> REGEX, there are other options you can consider:

- Named entities
- Dictionaries
- Divide a column in separate elements

Using Named Entity Recognition with EDM

What is NER?

- Algorithmic classifiers that detect specific types of entities:
  - All full names
  - All physical locations
  - Physical locations in different countries
  - "All last names" and others on the roadmap
  - Etc.
- Match based on format, context and internal dictionaries.

NER is not often used alone. Typically used to enhance accuracy of rules that look for PII

- e.g. an SSN *and someone's name*

Can also be used with EDM as a trigger SIT for the corresponding primary element, e.g. to match a "Full Name" column.

Main limitation: content like names and addresses may be written in a different way than how NER captures the match. E.g. it might capture a full address including province and country, while your table might include only the street address.

Using Dictionaries with EDM

Dictionaries? Isn't EDM better?

	Dictionaries	EDM
Table size	~100K rows (1MB compressed text)	100M rows
Accuracy	Single value match	Multi-column match
Case sensitivity	Case insensitive	Case sensitive or insensitive
Privacy	Need to upload the data to Microsoft	Only need to supply salted hashes of data
Max detection rate	Not limited	100s per second

EDM SIT triggered by a dictionary: the best of both worlds!

- Use dictionary to find candidates, use EDM to find the exact matches to combined columns.

But what about dictionary sizes? What if I have millions of records to protect?

- You might have 100M customers, but there aren't 100M last names.

First name	Last name	SSN
Kye	Reese	111-11-1111
John	Connor	983-87-3339
SARAH	CONNOR	292-09-2866
Peter	Silberman	283-33-2114
Ed	Traxler	091-22-9384

Deduplicate

Last name
Reese
Connor
Silberman
Traxler

Upload as dictionary

All last names
Connor
Reese
Silberman
Traxler

Hash and upload

Firstname	Lastname	SSN
56f6e20696e2	6f6e2069634	0636f6d70617
4b177e0db1d	82517c9053d	9e7280c96dd
ebc4103dda7	6e206f662073	580e00857dd
56f6e20696e2	6f6e2069634	0636f6d70617

Pro tip: sort the dictionary before upload (makes compression more efficient).

Bonus tip: You might also want to include additional conditions in the dictionary-based SIT.

(Very) advanced scenario: Parsing/splitting a column

What if you have to detect complex elements such as street addresses?
E.g. "29833 NE 29th St., Redmond, WA, 98052, USA"
Problems:

- Too many different addresses to fit in a dictionary
- Address might not be spelled out that same way in a document

Solution:

- Parse the column in parts
- Define SITs (dictionary or regex) for each
- Define EDM columns for each based on the SIT

E.g.:

29833	→	Street number regex-based SIT
NE 29th St.	→	Street name dictionary-based SIT (requires a match to street number and city or ZIP nearby)
Redmond	→	City name dictionary-based SIT
98052	→	Zip code regex-based SIT

Then create an EDM SIT pattern for "Customer street address" that uses the street name as primary evidence and a combination of the others plus some other PII as secondary (e.g. street name and number plus first name).

Structured process for defining SIT patterns

- Start by listing all the possible combinations of the fields in a table.
  - Hide all those that have few valid values (e.g. gender, “marital status”, etc.) since they don’t add value to the detection.
  - Mark all combinations that are of interest (example below is simplified, complex conditions might require more nuanced markings).
- Color code those rows and columns that have a structured, easy to detect format, that can be detected using a SIT.
  - Any combination that involves a colored row or column, can use that field as a primary element. You can remove any column that only has marks in colored cells.
- If there are remaining combinations that don’t involve a colored row or column, choose the row that is simplest to identify via a dictionary or NER and flag them as well.
  - Mark all the combinations involved as covered by using that column as primary element.
- Repeat the last step until there are no combinations not covered by a primary element that can be detected via a SIT.

	Last name	First name	National ID	Account #	Drivers license	Street Address	Date of birth
Last name			X	X	X	X	X
First name			X			X	
National ID	X	X		X	X	X	X
Account number	X		X		X	X	X
Drivers license	X		X	X		X	X
Street Address	X	X	X	X	X		
Date of birth	X		X	X	X		

Multi-token additional evidence columns

- Situation example : I created an EDM SIT that uses "company name" as additional evidence, and some rows don't match.
- Cause: by default, EDM matches additional evidence against *individual words* in the text surrounding a primary element match
  - So it can't match "multi-word" terms (e.g. compound last names, street names, phone numbers with spaces, etc.)
  - E.g.: if the street name column for the matched person includes "**Calder Cyn Rd.**" and the text in a document includes "**Sarah J. Connor lives in 309 Calder Cyn Rd.**", the system will compare the hashes of "**lives**", "**in**", "**309**", "**Calder**", "**Cyn**" and "**Rd.**" against the hash of "Calder Cyn Rd." and none of them will produce a match.
- Solution: assign a SIT to the additional evidence columns when you refer to them in the EDM SIT pattern, just like when you do it for primary evidence!
- How? That's not in the UI!
  - Indeed, but coming soon.
  - Today, you have to edit the XML for the EDM SIT, e.g.:

```
<ExactMatch id="cb664342-1b60-4451-a98f-68b378ce6f62" patternsProximity="300" dataStore="T-800" >
    <Pattern confidenceLevel="75">
        <idMatch matches="SSN" classification="U.S. Social Security Number (SSN)" />
            <Any minMatches="1" maxMatches="3">
                <match matches="Lastname" />
                <match matches="AddressStreet" classification="Street Names Dictionary"/>
                <match matches="Phone" />
            </Any>
    </Pattern>
</ExactMatch>
```

See https://docs.microsoft.com/en-us/microsoft-365/compliance/sit-modify-a-custom-sensitive-information-type-in-powershell for details.

Be aware: throttling limits *do not* apply to this SIT, so the SIT can be more "relaxed".

EDM Special scenarios

Specific cases of columns as primary evidence

We will discuss:

- Full names
- Last names
- Street addresses
- Account number / Medical Record Numbers with mixed patterns
- Date of birth
- Devices

Full names as primary element

Main challenge: names can look like regular words, and can have two, three, four or more words, some punctuation and even symbols in them.

Don't use a regular expression, e.g. "[A-Z][a-z]+([A-Z][a-z]+)"

- It will cause lots of false positives and result in throttling and missed detections.
- Once a regular expression matches some text, it won't use it again for further matches, for example the regex above applied to the phrase "**Patient John Smith was examined by Dr. Peter Robinson**" will match **Patient John** and **Peter Robinson**, but not **John Smith**.

Consider first: if you can use another column to use as primary. Most often, names only need to be detected in combination with other "harder" identifiers such as SSN or account numbers.

If you must use full names as primary evidence, evaluate using the "All Full Names" Named Entity Recognition classifier, it's the easiest option.

Limitations:

- Might not detect all names, depending on context, syntax and how uncommon a name may be, so there's some risk of false negatives.
- Some languages are not yet supported.
- Might detect a name in a different format than what's entered in the table (e.g. Smith, John instead of John Smith).

Alternative: divide full name column in first and last names and use dictionaries as we will discuss later.

Note: use the same techniques if you need to use full names as secondary elements in an EDM pattern.

Last names as primary element

Main challenge: Compound or multiple last names make detection with a regex hard.
ALSO don't use a regular expression, e.g. "[A-Z][a-z]+([A-Z][a-z]+)"

Same recommendation as for full names: avoid if possible.

NER for All Last Names will be available soon, with similar caveats as for All Full Names.

Most effective option: use the de-duplicated dictionary-based SIT approach suggested before:

First name	Last name	SSN
Kye	Reese	111-11-1111
John	Connor	983-87-3339
SARAH	CONNOR	292-09-2866
Peter	Silberman	283-33-2114
Ed	Traxler	091-22-9384

Deduplicate

Last name
Reese
Connor
Silberman
Traxler

Upload as dictionary

All last names
Connor
Reese
Silberman
Traxler

Hash and upload

Firstname	Lastname	SSN
56f6e20696e2	6f6e2069634	0636f6d70617
4b177e0db1d	82517c9053d	9e7280c96dd
ebc4103dda7	6e206f662073	580e00857dd
56f6e20696e2	6f6e2069634	0636f6d70617
4b177e0db1d	82517c9053d	9e7280c96dd

Real world example: hospital patient list with over ten million patients.
After de-duplication, less than 100K unique last names (including a few typos)
Sorted and compressed it used 260KB, fitting well within a keyword dictionary.
Also consider same technique for using last names as additional evidence.

Street addresses as primary element

Main challenges:

Street addresses may involve different formats across the world, or even within a country

Street addresses may be written differently (e.g. St., vs Street, Northeast vs. NE)

Street addresses may or may not include things like ZIP, city, state, etc.

Consider using All Physical Addresses (or country-specific physical addresses) NER classifier if addresses are consistently written in full.

Might consider using a complex regular expression if your country is not supported by NER, but beware of same challenges with partial addresses, addresses typed with different elements, etc.

Most effective method if you must use physical addresses as a primary element: use the parse/split method described before:

29833	→	Street number regex-based SIT and Street number column in schema
NE 29th St.	→	Street name dictionary-based SIT and Street name column in schema
Redmond	→	Cities dictionary-based SIT and City column in schema
98052	→	Zip code regex-based SIT and ZIP code column in schema

(parsing of the address column to separate in multiple columns can be scripted without much trouble)

Then use an EDM pattern that requires street name as the primary element and the other columns as secondary elements within e.g. 50 characters.

Note: some "street names" may be common words, so to avoid triggering too frequently, add requirements for matches to the street number, city and/or ZIP code regex or dictionary matches nearby so the SIT only matches likely addresses.

MRN, Account #, custom ID as primary elements

Typically composed of numbers and letters, or just numbers.

Challenges:

- Low numbers sometimes valid and may cause false positives. E.g. some hospitals started with MRN #1.
- Mixed formats: due to M&A, many formats may have accumulated over time.

Don't: use a regex like \w+, or even \b[A-Za-z0-9]{4,10}\b, will lead to throttling.

First impulse is to say "there's no pattern", but there's *always* a pattern.

Best approach: divide and conquer.

Take the first number and define a pattern that matches it. Filter the table to exclude values that match that regex (sample script in slide notes) and repeat the process, joining each regex to the previous one with a "|", until all values are covered.

E.g.:

03184383044
F1038800841
194343313443
334494388444
314331383444
14148809841
134901111441441
F4148319443

=

\b\d{11}\b
+
F1038800841
194343313443
334494388444
314331383444
134901111441441
F4148319443

=

\b\d{11}|F\d{10}\b
+
194343313443
334494388444
134901111441441

=

\b\d{11,12}|F\d{10}\b
+
134901111441441

=

\b\d{11,12}|\d{14}|F\d{10}\b

Additonal consideration: if some of the patterns are too "easy" (e.g. Four digit numbers), split them out and require either specific values from a dictionary (if few valid ones) or additional evidence (e.g. Keywords required for any MRN under 10000).

Date of birth as primary element

Don’t.

Date of birth as primary element

Seriously, don't.

Why not dates?

- It's not PII unless it is attached to a person. 12/20/1965 is someone's birthday, am I leaking something here if I don't say whose?
- Dates are everywhere. The average email has 20 of them (including headers). You WILL get throttled.
- They are written in many different formats. You will get false negatives.
- Same birth date is shared by hundreds of thousands of people. EDM processing will timeout and miss matches.

Use anything else that's more uniquely identifying for primary element, and only use birth dates as secondary elements if needed.

What about using birth date as additional element then? I need a SIT, right?

You *CAN* use a regex for birthdays as additional evidence, but be sure to exclude dates that are obviously not birth dates (dates in the future or too recent if you aren't a hospital, dates followed by timestamps, dates in the Sent row of an email, etc.).

E.g. `\b([012]?\d|3(0|1))(\/|-|\.)([012]?\d|3(0|1))(\/|-|\.)(19\d{2}|20(0\d)|\d{2})(?! \d{1,2}:\d{2}(:\d{2})?)\b`

Or

```
(?<!^Sent: (Monday|Tuesday|Wednesday|Thursday|Friday|Saturday|Sunday),
)(January|February|March|April|May|June|July|August|September|October|November|December) (1|2|3)?\d(,| of)? (1|2)\d{3}(?:
\d{1,2}:\d{1,2}( (AM|PM))?)?$
```

BUT keep in mind dates might be entered in different formats in documents. You might have to include multiple versions as additional columns, e.g. D/MM/YYYY, D/M/YY, YYYY/MM/DD, etc.

Device info as primary element

You can use EDM for device names, IDs, IP address, MAC addresses, etc.

But all those are tricky, since they are part of every email header!

Also, server names can be common words.

Solution:

1) Exclude all names that are part of a message header, e.g. use a regex that excludes matches before "MIME-Version" using a negative lookahead.
2) Exclude all names in special lines like "Received: from" using a negative lookbehind.
3) Exclude all known email server names using a dictionary as a negative condition (use MaxMatches=0 in the SIT XML)
4) If device names follow a strict pattern, use a regex.
5) For server names, use a dictionary, unless you have hundreds of thousands of servers they should fit.
6) For IP addresses, MAC addresses, etc., you can use a regex, but make sure to exclude all those in email headers.
7) If using a pattern for something like an unformatted MAC address or IP address, make sure you properly surround your regex with \b, to avoid detecting substrings of things like GUIDs.
8) Test the SIT exhaustively, and add exclusions to your logic as necessary to reduce unintended matches before you configure an EDM SIT based on it.

What if none of those things work?

- What if I can't use a regex?
- What if there's no NER for what I want to detect?
- What if I have too many values to put in a dictionary?

Get creative.

E.g.

- Write a script that looks for the longest or most infrequent word in each string in the table, de-duplicate them, and put them in a dictionary and in a separate column, use *that* as the primary or secondary element instead of the full value. If not enough, repeat the process for the *second* longest or most infrequent word and use as secondary evidence to reduce FPs.

This can be hard, but there are very few cases where there is no viable solution.

If stuck, post your question on Yammer. The community should be able to help.

Appendix: collateral reading (if you are masochist)

- Sensitive info type definitions: https://aka.ms/sensitiveinfotypes
- Sensitive info type XML syntax for manual edit of SITs: https://docs.microsoft.com/en-us/microsoft-365/compliance/sit-get-started-exact-data-match-create-rule-package
- Configuring EDM: https://docs.microsoft.com/en-us/microsoft-365/compliance/sit-get-started-exact-data-match-based-sits-overview
- Troubleshooting EDM: https://docs.microsoft.com/en-us/microsoft-365/compliance/sit-get-started-exact-data-match-test
- Third party regular expression resources:

  https://regexr.com/ (great tool for learning by trial and error, though it doesn't strictly support the Microsoft syntax)
  http://regexstorm.net/tester (great for troubleshooting, supports the exact Microsoft implementation of regex)
  http://www.rexegg.com/ (extremely thorough regex tutorial)