DECODE2LOD: Connecting the DECODE Database with the Linked Open Data Cloud

Cosimo Palma
University of Naples "L'Orientale"
University of Pisa, Italy
cosimo.palma@phd.unipi.it

Beáta Megyesi
Department of Linguistics
Stockholm University, Sweden
beata.megyesi@ling.su.se

Abstract

This paper presents a novel approach to enhancing the analytical power and interoperability of historical cryptology data by transforming the DECODE database into a Linked Open Data (LOD) resource. We introduce a methodology for modeling encrypted historical documents and cipher keys as a knowledge graph, encompassing ontology development, data transformation, and SPARQL-based querying. This integration enables complex queries across domains, encourages collaboration beyond cryptology, and aligns DECODE with broader efforts in digital humanities and open science. By bridging historical cryptology with LOD principles, we offer a scalable framework for enriching specialized research databases through semantic technologies.

1 Introduction

Historical cryptology, like all scientific fields, depends on robust research infrastructure. This infrastructure typically includes systematic, well-curated collections of diverse data sources described through metadata, alongside tools designed to process and analyze the data. The DECODE database (Megyesi et al., 2019; Héder and Megyesi, 2022), with its repository of nearly 10,000 records at the time of writing, has become the de facto standard resource for historical cryptology research. Each record in DECODE—whether a ciphertext, cipher key, or manual—is enriched with extensive metadata describing the source's current location, origin, content, and additional analyses. As a searchable, evolving monitor corpus, DECODE continuously incorporates new data, facilitating numerous studies in historical cryptology.

Beyond its use by historical cryptologists, the DECODE database holds significant potential for researchers in adjacent fields, such as Digital History. However, to fully leverage the database's richness and to enable more sophisticated, granular analysis, a technological advancement in its integration is necessary. This paper addresses this need by proposing the integration of the DECODE database with Linked Open Data (LOD), enabling advanced querying, fostering interdisciplinary research, and supporting interoperability with external datasets.

2 Digital History and Linked Open Data

The decipherment of encrypted manuscripts has often provided new insights into history, reshaping our understanding of the past. A recent example is the decryption of Mary Stuart's encrypted letters (Lasry et al., 2023), which offer remarkable glimpses into her life as a prisoner under the watch of the Earl of Shrewsbury. Furthermore, each deciphered text must be examined within its historical and cultural framework, taking into account the time period, geographic location, and sociopolitical context.

Interdisciplinary collaboration among scholars from different fields enriches the decipherment process in multiple ways. Historians and linguists, for instance, can provide cryptologists with crucial contextual insights and help interpret the paratextual elements of ciphers (Megyesi et al., 2020). This is particularly important when encrypted texts go beyond a simple sequence of characters and take the form of complex artifacts embedded within a specific historical and cultural setting, as discussed in (Palma, 2023).

Digital history is increasingly dependent on computational tools and methodologies for analyzing historical data, often utilizing structured databases, text mining, and topic modeling to generate informative visualizations. Among these

Proceedings of the 8th International Conference on Historical Cryptology, pages 111-121
Poznań, Poland, 16-18 June, 2025

tools, LOD has emerged as a key approach for connecting disparate datasets through semantic relationships, thereby enhancing both discoverability and interoperability.

One of LOD's strengths is its support for federated queries, which allow users to retrieve and integrate information from multiple distributed databases in a single query, eliminating the need for centralized data storage. This capability improves data granularity and accessibility, enabling researchers to explore interconnected historical sources more effectively. By facilitating seamless cross-database exploration, LOD makes historical research more interactive, integrative, and comprehensive.

The concept of Linked Data refers to a set of best practices for publishing structured data on the Web. These principles were introduced by Tim Berners-Lee in his Linked Data design issue note[1], and reformulated through the FAIR principles– *Findability*, *Accessibility*[2], *Interoperability*, and *Reusability*—-which aim to ensure that data is structured, discoverable, and reusable. To align with these principles, the dataset has been modeled using Uniform Resource Identifiers (URIs) for each record, ensuring unique identification and, where possible, aligning properties with existing ontologies to promote interoperability.

Despite DECODE's extensive metadata framework and its proven utility, the database had yet to be integrated into the LOD cloud, thus enabling for more complex and scalable analyses. By aligning DECODE with LOD principles, researchers can unlock deeper insights into encrypted sources and connect historical cryptology with broader datasets in cultural heritage and other domains. To address this gap, this study pursues the following objectives:

- Creating an **ontology** for DECODE that mirrors its existing metadata.
- **Populating** the ontology with DECODE's current dataset, ensuring semantic alignment.
- Designing and demonstrating example **SPARQL queries** that showcase the enhanced analytical capabilities enabled by LOD integration.

The following questions guide this research in demonstrating the relevance and added value of the proposed approach:

- How does this data representation improve upon the existing metadata framework?
- What new insights and research opportunities can be derived from SPARQL queries applied to the LOD-integrated DECODE database?

By addressing these objectives and research questions, this study seeks to advance the field of historical cryptology and establish a model for integrating domain-specific databases, particularly those that do not conform to tabular or relational data structures—into the broader LOD ecosystem.

3 From Relational to Graph Databases: Modeling DECODE for the Semantic Web

A brief account on how to model a database containing a collection of ciphers has been already provided by Bonavoglia (Bonavoglia, 2023).

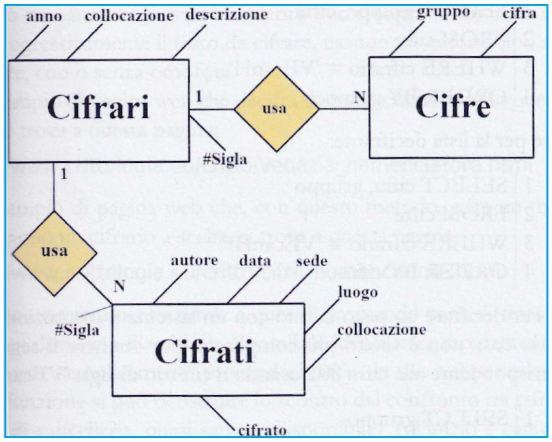

Figure 1: An overview of tables and properties in the relational database proposed by Bonavoglia (Bonavoglia, 2023).

His model is based on a relational database in which entries are stored in tables linked by specific keys. The typical query language for relational databases is SQL, which retrieves information by manipulating the tables in which information is stored. For example, in Figure 1, "Encryption method" (*Cifrari*), "Ciphers" (*Cifre*) and "Cyphertexts"/records (*Cifrati*) would be represented as *tables*, and their properties as columns. Querying common elements between encryptions (top-right box) and encrypted texts (bottom box) would require running a query that performs a union over the encryption methods table.

[1] `https://www.w3.org/wiki/LinkedData`

[2] Currently, the Knowledge Graph developed by the authors is not published online, i.e. the related SPARQL query endpoint is not hosted on any web domain.

Relational databases have traditionally been favored for efficiently capturing relationships between entries, but they fall short when it comes to representing the detailed properties of those entries. In contrast, graph-based databases not only capture these relationships, but also define and store properties for each node. This allows them to represent complex, interconnected data more naturally and flexibly than relational databases, leading to higher performance and a more intuitive data model for many modern applications.

Although graph databases offer advantages in performance, flexibility, and scalability, these benefits are minimal for a medium-to-small database like DECODE. However, their ability to support interoperability with other knowledge graphs—an essential feature in the rapidly evolving field of Digital Humanities—is uniquely facilitated within a LOD framework.

The literature presents various methods for converting datasets into Resource Description Framework (RDF)[3] format (Klyne and Carroll, 2004), regardless of whether the source is a structured spreadsheet (Fiorelli et al., 2015), a natural language description (di Buono et al., 2014), or a more heterogeneous data collection (Dimou et al., 2014).

To ensure compatibility and semantic integration, the DECODE Knowledge Graph is aligned with CIDOC-CRM (Doerr, 2003), the de facto standard ontology for Cultural Heritage, as well as with more general ontologies such as Schema.org (Guha et al., 2016) and Dublin Core (Baker, 2000; Weibel and Koch, 1997).

4 Building DECODE2LOD

To make the DECODE database LOD-compatible[4], we present a workflow and describe the data collection process along with the steps involved in ontology development and semantic transformation. The workflow is illustrated in Figure 2.

Data acquisition from the current DECODE database consists of several interconnected functions that systematically retrieve and process records from a REST API[5]. The ontology population process, which structures the acquired data within a defined schema, includes three key components: 1) ontology initialization, 2) data harvesting and ingestion, and 3) semantic transformation. Each of these components are described in detail below.

4.1 DECODE to LOD Conversion

The conversion of the DECODE2LOD begins by initializing an RDF graph structure with a predefined name space (https://de-crypt.org/r/), incorporating an ontology from a Turtle (TTL) file. The Ontology has been previously crafted manually to both mirror the structure and metadata of a DECODE record shown in the record view of the database, as illustrated in Figure 3.

Initially, the system establishes a base URL for API communication. The primary function, ”fetch records”, constructs API endpoint URLs by combining the base URL with specific table identifiers and handles HTTP GET requests with pagination parameters. This function returns JSON-formatted responses or raises appropriate exceptions upon failure.

To gather the complete dataset, the ”fetch all records” function implements an iterative pagination mechanism, collecting all available records page by page.

Individual record details are accessed through the ”fetch specific record” function, which constructs endpoints for specific record IDs and retrieves detailed information for each record. The ”fetch all record details” function extends this capability by processing collections of records, systematically retrieving detailed information for each valid record ID while gracefully handling missing or invalid identifiers.

Data persistence is managed through the ”save to json” function, which serializes the collected data into JSON[6] format and writes it to specified

[3]As described in `http://w3.org/RDF/`, RDF is a standard model for data interchange on the Web. It facilitates data integration even when underlying schemas differ and supports schema evolution over time without requiring modifications from all data consumers.

[4]The codebase for this operation can be retrieved at `https://github.com/Glottocrisio/Decode2LOD/tree/main`

[5]REST provides a set of architectural constraints that, when applied as a whole, emphasizes scalability of component interactions, generality of interfaces, independent deployment of components, and intermediary components to reduce interaction latency, enforce security, and encapsulate legacy systems (Fielding and Taylor, 2002). The DECODE API can be accessed at: `https://de-crypt.org/decrypt-web/api`

[6]JSON (JavaScript Object Notation, pronounced /desn/ or /desn/) is an open standard file format and data interchange format that uses human-readable text to store and transmit data objects consisting of name–value pairs and arrays (or other serializable values). It is a commonly used data format

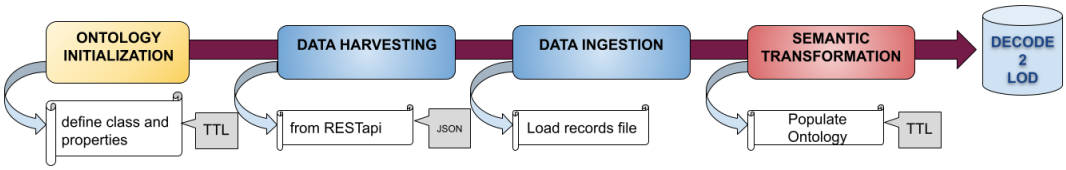

Figure 2: The workflow of converting DECODE to LOD.

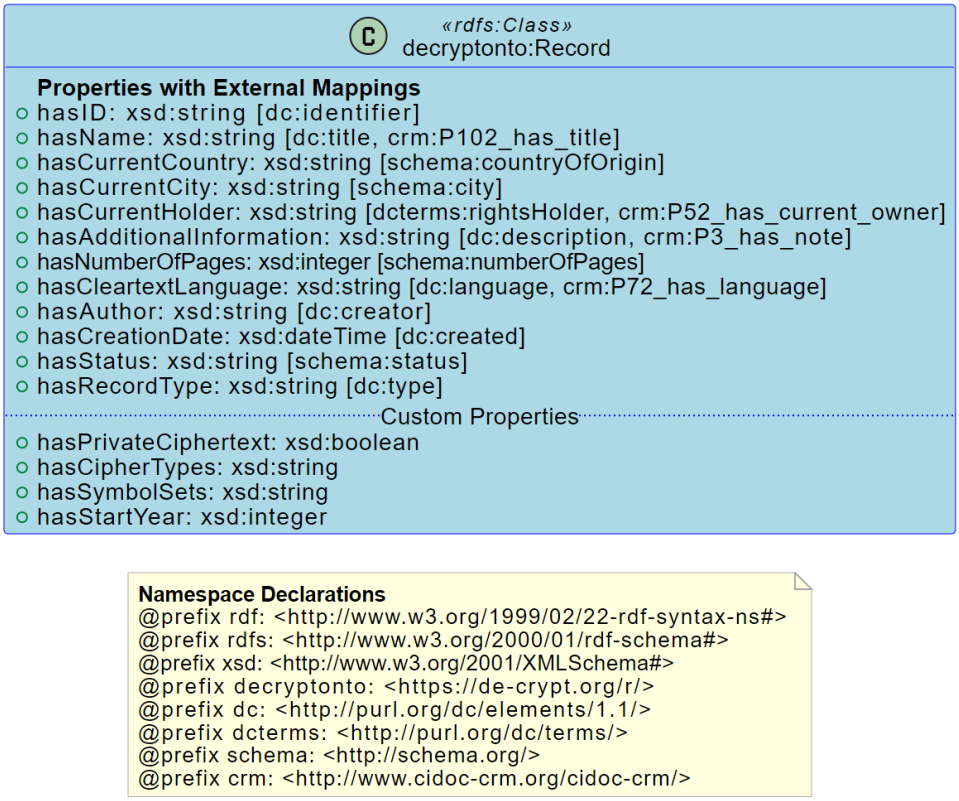

Figure 3: A specification of ”Record”, the only class in the Decrypt Ontology.

files. The main routine orchestrates these components by defining target tables and output files, coordinating the collection of both summary and detailed records, and ensuring proper error handling throughout the execution process.

The entire system implements error handling at multiple levels to manage potential failures in API communication, data processing, or file operations, ensuring robust operation in production environments.

The data acquisition phase implements a robust JSON parsing mechanism with built-in encoding fallback. This component initially attempts to parse the input file using UTF-8 encoding, and if unsuccessful, automatically retries with UTF-8-SIG encoding, ensuring compatibility with various text encodings that may be present in real-world datasets.

The core semantic transformation process maps structured data to ontological concepts through a comprehensive property mapping system.

Properties within the ontology are defined as *rdf:Property* types and are constrained to operate exclusively within the *decryptonto:Record* domain. Each property must explicitly declare its range, which is restricted to the following XML Schema Definition (XSD) data types: *string, integer, dateTime*, or *boolean*. Documentation

with diverse uses in electronic data interchange, including that of web applications with servers. (*Source: Wikipedia*)

for all properties is eventually provided through *rdfs:comment* annotations. Additionally, properties may be semantically linked to external ontologies through *rdfs:sameAs* mappings, enabling interoperability across different knowledge systems.

This mapping framework establishes precise correspondences between source data attributes and ontological predicates, while simultaneously defining appropriate XML Schema data types for literal values. The transformation process iterates through each record, creating unique URIs for individual entities and establishing their type relationships within the ontological framework. For each property in the mapping dictionary, the system performs datatype-specific transformations, handling special cases such as Boolean value normalization, Integer conversion, and DateTime parsing with ISO 8601 compliance.

Upon completion, the populated ontology is serialized in Turtle (.ttl) format, producing a semantically enriched representation of the original dataset.

4.2 Dataset Blueprint

The resulting DECODE Knowledge Graph comprises 25,970 nodes and 115,303 edges. The nodes include both the record themselves and the (both *literal-* and *object-*) values of every property, in this case displayed as edges (or relationships).

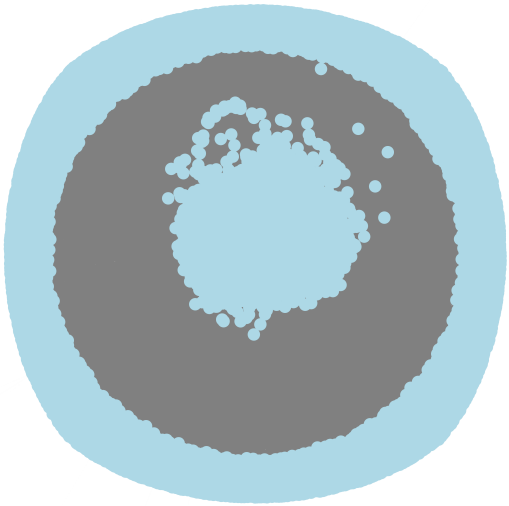

Figure 4: A distant view of the DECODE-LOD knowledge graph.

The average node centrality, calculated using the formula

$$\text{Average Centrality} = \frac{2 \times \text{Total Relationships}}{\text{Total Entities}}$$

is 8.88, meaning that each entity is, on average, connected to 9 other entities. Given that the dataset contains 9,390 records, an additional 16,580 nodes represent property values. As illustrated in Figure 4, individual records (represented by blue dots in the center) are not directly linked to one another but are instead connected to various properties (the dense cluster of blue dots at the periphery). The dark-grey zone represents the extensive network of relationships originating from these records—though visually indistinguishable due to graphical limitations. To enhance interconnections among record-entities, several strategies can be implemented, as it is discussed in Section 6.

5 Querying the Knowledge Graph

Visualizing and querying a knowledge graph are two distinct yet complementary processes. While visualization helps users explore the structure and relationships within the graph, querying enables the extraction of specific insights through a structured search.

5.1 Executing Queries

The Turtle file generated in the previous workflow cannot be used directly for querying. Although various online tools facilitate knowledge graph visualization[7], executing queries requires downloading and running appropriate software locally. One of the most popular choices for this purpose is Apache Jena Fuseki[8], known for its balance between user-friendliness and performance. As illustrated in Figure 5, setting up Apache Jena Fuseki involves downloading and extracting the package into a local directory, then running the ”fuseki-server” Windows batch file.

Once the server is running, the URL displayed in the command prompt window (highlighted in red in the figure) must be copied and pasted into a web browser. From there, the Turtle file containing the Knowledge Graph can be loaded, allowing users to execute SPARQL queries through the built-in SPARQL endpoint . Once the Turtle file is successfully loaded, users can dynamically explore and analyze the Knowledge Graph, lever-

[7]Figure 3 was generated using `https://semantechs.co.uk/turtle-editor-viewer/`, while Figure 4 was produced using `https://issemantic.net/rdf-visualizer`.

[8]Available for download at `https://jena.apache.org/download/index.cgi`.

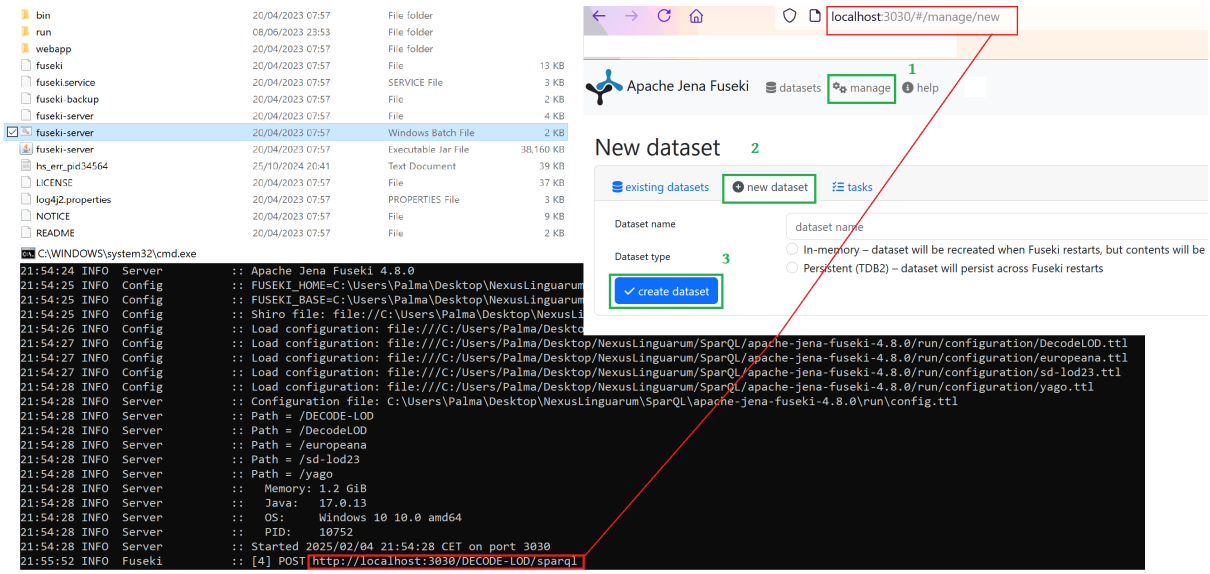

Figure 5: An overview of the Apache Jena Fuseki environment. The selection of the Windows Batch File will open a prompt featuring the number of port that needs to be added to ”localhost” in the browser search bar.

aging SPARQL queries to extract meaningful insights.

5.2 Example Queries

We illustrate the capabilities of the DECODE knowledge graph by presenting a series of advanced queries designed to explore historical cryptographic records. The primary user persona considered here is the *Digital Historian*, an historian using digital tools for exploring and gathering data about historical events. By leveraging SPARQL—the query language for RDF databases—we demonstrate how DECODE2LOD can facilitate nuanced investigations that would be challenging or impossible using traditional database search functionalities.

Some queries require federated search, where information from DECODE2LOD is cross-referenced with external knowledge bases such as DBpedia[9], a structured knowledge base derived from Wikipedia. This approach enables researchers to enrich their analyses with additional historical and contextual information. For example, the historian might want to *retrieve all unresolved ciphers from any European country that was also involved in a war during the seventeenth century.* The answer to the research question requires a federated query where the encrypted records are retrieved from the DECODE2LOD database, and are cross-referenced with DBpedia to gather additional historical context, such as wars in which a country was involved. The query is shown below for this particular example:

```
PREFIX rdf:
<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs:
    <http://www.w3.org/2000/01/rdf-schema#>
PREFIX decryptonto: <https://de-crypt.org/r/>
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX dbr: <http://dbpedia.org/resource/>
PREFIX dbp: <http://dbpedia.org/property/>
PREFIX dcterms: <http://purl.org/dc/terms/>
PREFIX dbc:
    <http://dbpedia.org/resource/Category:>
PREFIX xsd:
    <http://www.w3.org/2001/XMLSchema#>

SELECT DISTINCT ?cipherName ?country ?warName
    ?warYear
WHERE {
  # Query the DECRYPT ontology for unresolved
    ciphers
  ?cipher rdf:type decryptonto:Record ;
          decryptonto:hasName ?cipherName ;
          decryptonto:hasCurrentCountry
    ?country ;
          decryptonto:hasStatus ?status .

  # Filter for unresolved ciphers
  FILTER(?status != "4" && ?status != 4)

  # Prepare country variables for matching
```

[9] https://www.dbpedia.org/

```
  BIND(LCASE(?country) AS ?countryLower)
  BIND(REPLACE(STR(?country), " ", "_") AS
    ?countryNormalized)
  BIND(CONCAT("http://dbpedia.org/resource/",
    ?countryNormalized) AS ?countryResource)
  BIND(CONCAT("http://dbpedia.org/resource/
  Category:Battles_in_", ?countryNormalized)
    AS ?battleInCategory)
  BIND(CONCAT("http://dbpedia.org/resource/
  Category:Battles_involving_",
    ?countryNormalized) AS
    ?battleInvolvingCategory)

  # Federated query to DBpedia for wars
  SERVICE <https://dbpedia.org/sparql> {
    # Get the war information
    ?war rdf:type dbo:MilitaryConflict ;
         rdfs:label ?warName .

    # Try multiple date formats
    OPTIONAL {
      ?war dbo:date ?date .
      BIND(YEAR(?date) AS ?warYear)
    }

    # If no date directly available, try
    start date
    OPTIONAL {
      ?war dbo:startDate ?startDate .
      BIND(YEAR(?startDate) AS ?warYear)
    }

    # Only use date if available and in 17th
    century
    FILTER(BOUND(?warYear) && ?warYear >=
    1600 && ?warYear < 1700)
    FILTER(LANG(?warName) = "en")

    # Connection to the country - using
    multiple approaches
    {
      # Option 1: Direct link to country
      ?war dbo:wikiPageWikiLink
    ?countryResource
    } UNION {
      # Option 2: Combatant contains country
    name
      ?war dbo:combatant ?combatant .
      FILTER(CONTAINS(LCASE(STR(?combatant)),
    ?countryLower))
    } UNION {
      # Option 3: Property combatant contains
    country name
      ?war dbp:combatant ?propCombatant .

    FILTER(CONTAINS(LCASE(STR(?propCombatant)),
    ?countryLower))
    } UNION {
      # Option 4: Place in country
      ?war dbo:place ?place .
      ?place rdfs:label ?placeLabel .
      FILTER(LANG(?placeLabel) = "en")
      FILTER(CONTAINS(LCASE(?placeLabel),
    ?countryLower))
    } UNION {
      # Option 5: Battles in country category
      ?war dcterms:subject ?battleInCategory
    } UNION {
      # Option 6: Battles involving country
    category
      ?war dcterms:subject
    ?battleInvolvingCategory
    }
  }
}
ORDER BY ?country ?warYear
LIMIT 100
```

The output of the query is shown in Figure 6, listing war names and starting years to illustrate the descriptive capabilities of the executed query. Additional information, such as the locations and key participants in each event, could also be included. The answer to the research question posed by the query appears in the first column, *cipherName*. This column was chosen over displaying full URIs due to space constraints; the URIs, however, remain accessible via clickable links that redirect to the corresponding DECODE record pages.

The DECODE database includes two built-in search functionalities: the *Advanced Search* and the *Query Builder*. The Advanced Search, shown at the left of Figure 7, corresponds to a natural language query such as: ”Retrieve all records that have...”—with specific properties and values inserted by the user. The Query Builder, displayed at the right of Figure 7, offers a more refined and flexible search experience. Unlike Advanced Search, which implicitly uses the AND operator and assumes equality conditions, the Query Builder allows users to define more complex queries using logical operators such as OR, as well as a broader set of conditions like ”contains,” ”ends with,” and their respective negations.

In terms of expressive power, the Query Builder more closely aligns with the SPARQL query-endpoint, which not only allows for precise data retrieval but also provides control over the output format. SPARQL enables users to refine result sets by defining specific entities to be displayed, incorporating optional statements, and setting limits on the number of rows. This is achieved using constructs such as DISTINCT, REDUCED, ORDER BY, LIMIT, OFFSET, and GROUP BY. Moreover, SPARQL supports data manipulation functions (e.g., LANG, DATATYPE, BOUND, IRI, URI, STRDT, STRLANG) and conditional expressions (e.g., IF, COALESCE, BNODE, CASE WHEN, THEN, ELSE). Some queries involve them alongside graph pattern matching (using BIND), making them more complex than those that can be constructed with the Query Builder. These queries

Table | Response | 100 results in 15.314 seconds | Simple view | Ellipse | Filter query results | Page size: 50

	cipherName	warName	warYear
1	BL_Add_MS_18983_001	"Battle of Nieuwpoort"@en	"1600"^^<http://www.w3.org/2001/XMLSchema#integer>
2	BL_Add_MS_18983_001	"Siege of San Andreas (1600)"@en	"1600"^^<http://www.w3.org/2001/XMLSchema#integer>
3	BL_Add_MS_18983_002	"Battle of Nieuwpoort"@en	"1600"^^<http://www.w3.org/2001/XMLSchema#integer>
4	BL_Add_MS_18983_002	"Siege of San Andreas (1600)"@en	"1600"^^<http://www.w3.org/2001/XMLSchema#integer>
5	BL_Add_MS_18983_003	"Battle of Nieuwpoort"@en	"1600"^^<http://www.w3.org/2001/XMLSchema#integer>
6	BL_Add_MS_18983_003	"Siege of San Andreas (1600)"@en	"1600"^^<http://www.w3.org/2001/XMLSchema#integer>

Figure 6: First six rows of query results for Query 3, showing battles in England from the 17th century and DECODE records of unsolved ciphers from the same period.

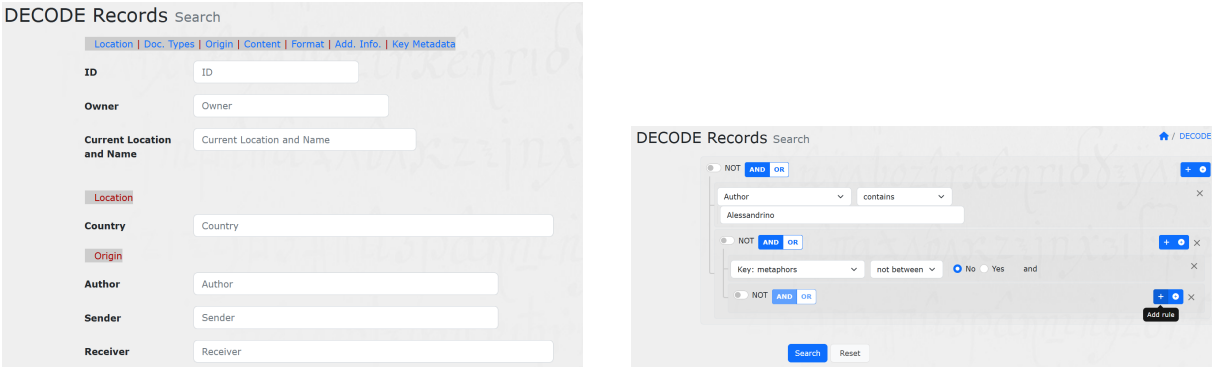

Figure 7: A snapshot of advanced search (left) and query builder (right) in the DECODE database.

support more advanced analyses, such as tracing the geographical spread of cipher usage over time or performing cross-cultural comparisons of cipher complexity—both of which require enhanced querying capabilities. Examples of these two query scenarios are provided in Appendix (7).

6 Limitations and Challenges

While the Query Builder offers a major advantage by allowing users—particularly those without experience in SPARQL or other query languages—to construct complex queries in an intuitive and user-friendly way, the dataset and underlying model presented here still face several limitations. These challenges currently hinder the system's immediate deployment and highlight the need for further development to improve its usability, interoperability, and completeness. To address these issues, several key enhancements are planned:

1. Enhancing Graph Density by Automated Inference.
2. Improving Interoperability and Wikidata Integration.
3. Integrating Large Language Models (LLMs) with Linked Open Data (LOD).

Enhancing Graph Density through Automated Inference This task focuses on creating new connections between existing nodes in the knowledge graph. When two records share similarities, relationships can be established using CIDOC-CRM properties such as P67 (refers to) or P130 (shows features of). Alternatively, a semantic rule layer can be manually defined to enable automated inference, enriching the graph with additional entities and relationships.

SPARQL serves not only as a query language for retrieving data but also as a means to manipulate the knowledge base by updating, deleting, or inserting nodes and relationships. Automated approaches can be employed to infer missing links or align the *decryptonto:hasAuthor* property with external sources like Wikipedia. Although many authors appear in both DECODE and Wikipedia, inconsistencies in naming conventions—such as "Cardinal Mazzarino" versus "Giulio Mazzarino"—necessitate further normalization before integration into DECODE2LOD can be achieved.

Improving Interoperability and Wikidata Integration Continuously aligning DECODE with Wikipedia and Wikidata for entities such as au-

thors, countries, years, and languages is essential not only for interoperability but also for enhancing its usability within the Digital History domain. For example, the author of the record shown in Figure 8 corresponds to the historical figure depicted in Figure 9.

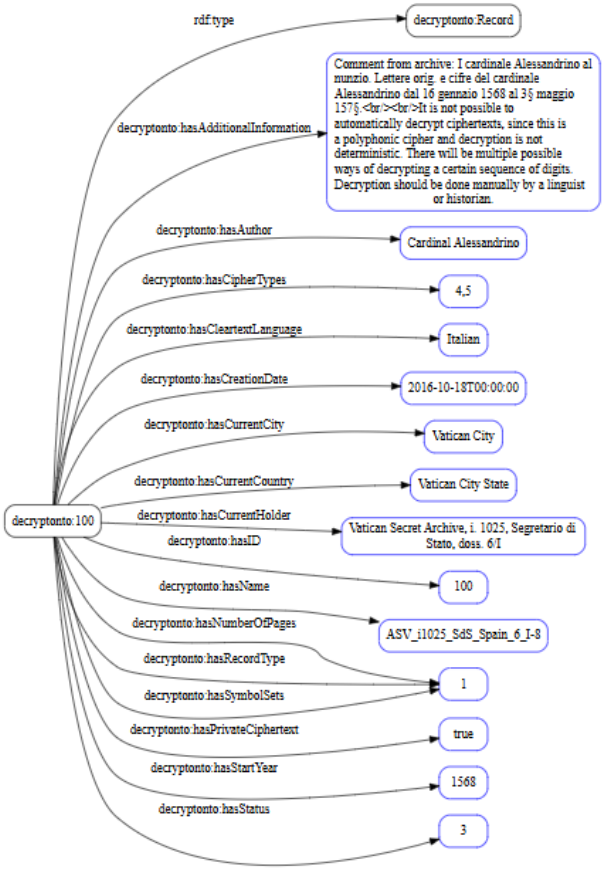

Figure 8: Visualization of a ciphertext from the DECODE database, whose author is Cardinal "Alessandrino", its cleartext language is Italian, and the cipher is stored at the Vatican city.

Figure 9: A portrait of Cardinal Michele Bonelli, known as "Alessandrino" (Source: *Wikipedia*)[11].

Wikipedia searches have also proven useful for disambiguating authors' nicknames. To improve clarity, a new property such as *decryptonto:alternativeName* could be introduced to link alternative names used for ciphertext authors.

Additionally, several key properties require alignment with their Wikidata equivalents, including:

- decryptonto:hasStartYear
- decryptonto:hasAuthor
- decryptonto:hasCleartextLanguage
- decryptonto:hasCreationDate
- decryptonto:hasCurrentCity
- decryptonto:hasCurrentCountry
- decryptonto:hasCurrentHolder

Integrating Large Language Models (LLMs) with Linked Open Data (LOD) While Pretrained Language Models (PLMs) often face challenges with accuracy and factual consistency—particularly due to their tendency to hallucinate information—they are remarkably effective at automatically generating SPARQL queries. This makes them especially valuable for improving accessibility and simplifying interaction with Linked Open Data (LOD) systems. Rather than viewing LLMs and LOD as competing technologies, they should be seen as complementary: LLMs enhance query formulation and user engagement, while LOD ensures structured, verifiable, and precise knowledge retrieval.

7 Concluding Remarks

In this paper, we presented the transformation of the DECODE database into a graph-based structure using Linked Open Data (LOD) to enhance its analytical potential. Unlike traditional relational databases, which rely on predefined table structures, a knowledge graph offers a more flexible and interconnected model, enabling complex queries that span multiple dimensions of historical cryptology. This transition not only improves data storage and retrieval but also facilitates advanced search techniques, such as semantic reasoning and federated queries. The improvements discussed in this paper aim to enhance both the technical robustness and practical accessibility of the historical cryptology ontology, making it a more powerful resource for digital humanities researchers.

However, deploying and maintaining a SPARQL endpoint on a web domain presents several challenges, particularly in terms of long-term maintenance. While these concerns may be minimal for the current dataset, they become

more significant as the dataset scales or faces high query loads. As more historical cryptographic records are digitized and integrated, the system must efficiently manage not only increased data volume but also complex relationship patterns and inference requirements.

Beyond storage capacity, scalability challenges also involve query optimization, cache management, and concurrent user access patterns. These technical constraints must be carefully balanced against the needs of digital humanities scholars, who often conduct exploratory analyses that generate resource-intensive queries. To ensure sustainable long-term operation, implementing strategies such as caching mechanisms, query rate limiting, and load balancing will be essential. These measures will help maintain system performance while preserving the analytical capabilities that make the graph-based approach valuable for historical cryptology research.

Acknowledgments

We gratefully acknowledge Mihály Héder for his invaluable advice and support, particularly during the technical implementation of the work presented here. We sincerely thank the anonymous reviewers for their insightful comments and constructive suggestions, which greatly improved the final version of this paper.

We also extend our thanks to the Department of Digital History (C²DH) at the University of Luxembourg, the Swedish Research Council (grant 2018-06074) for supporting the DECRYPT – Decryption of Historical Manuscripts project, and Riksbankens Jubileumsfond (grant M24-0028) for funding the DESCRYPT – Echoes of History: Analysis and Decipherment of Historical Writings project.

References

Thomas Baker. 2000. The Dublin core metadata initiative. *D-lib magazine*, 6(12):1082–9873.

Paolo Bonavoglia. 2023. *La crittografia della Repubblica di Venezia*. Aracne.

Maria Pia di Buono, Mario Monteleone, and Annibale Elia. 2014. How to populate ontologies: Computational linguistics applied to the cultural heritage domain. In Elisabeth Métais, Mathieu Roche, and Maguelonne Teisseire, editors, *Natural Language Processing and Information Systems*, volume 8455 of *Lecture Notes in Computer Science*, pages 55–66. Springer.

Anastasia Dimou, Miel Vander Sande, Pieter Colpaert, Ruben Verborgh, Erik Mannens, and Rik Van de Walle. 2014. RML: A generic language for integrated rdf mappings of heterogeneous data. In *Proceedings of the 7th Workshop on Linked Data on the Web*, LDOW '14. CEUR Workshop Proceedings.

Martin Doerr. 2003. The CIDOC CRM - An ontological approach to semantic interoperability of metadata. *AI Magazine*, 24(3).

Roy T. Fielding and Richard N. Taylor. 2002. Principled design of the modern Web architecture. *ACM Transactions on Internet Technology*, 2(2):115–150.

Manuel Fiorelli, Tiziano Lorenzetti, Maria Teresa Pazienza, Armando Stellato, and Andrea Turbati. 2015. Sheet2rdf: a flexible and dynamic spreadsheet import&lifting framework for rdf. In *Current Approaches in Applied Artificial Intelligence*, pages 131–140. Springer International Publishing.

Ramanathan V Guha, Dan Brickley, and Steve Macbeth. 2016. Schema. org: Evolution of structured data on the web. *Communications of the ACM*, 59(2):44–51.

Mihály Héder and Beáta Megyesi. 2022. The DECODE Database of Historical Ciphers and Keys: Version 2. In *Proceedings of the 5th International Conference on Historical Cryptology, HistoCrypt22*.

Graham Klyne and Jeremy J. Carroll. 2004. Resource Description Framework (RDF): Concepts and Abstract Syntax. *Journal of Web Semantics*, 2(1):9–29.

George Lasry, Norbert Biermann, and Satoshi Tomokiyo. 2023. Deciphering Mary Stuart's lost letters from 1578-1584. *Cryptologia*, 47(2):101–202.

Beáta Megyesi, Nils Blomqvist, and Eva Pettersson. 2019. The DECODE Database: Collection of Ciphers and Keys. In *Proceedings of the 2nd International Conference on Historical Cryptology, HistoCrypt19*, Mons, Belgium, June.

Beáta Megyesi, Bernhard Esslinger, Alicia Fornés, Nils Kopal, Benedek Láng, George Lasry, Karl de Leeuw, Eva Pettersson, Arno Wacker, and Michelle Waldispühl. 2020. Decryption of historical manuscripts: the DECRYPT project. *Cryptologia*, 44(6):545–559.

Cosimo Palma. 2023. Encrypted epigraphy - the case of a mysterious inscription in the Neapolitan church of Santa Maria La Nova. In *International Conference on Historical Cryptology*.

Stuart L Weibel and Traugott Koch. 1997. The Dublin core: a simple content description model for electronic resources. *Bulletin of the American Society for Information Science and Technology*, 24(1):9–11.

Appendix

Query 1. Geographical Spread of Cipher Usage Over Time

```
SELECT ?period ?region (COUNT(DISTINCT
    ?cipher) AS ?cipherCount)
WHERE {
  ?cipher decryptonto:hasStartYear ?year ;
          decryptonto:hasCurrentCountry
    ?country .

  # Define historical periods
  BIND(
    IF(?year < 1500, "Medieval",
      IF(?year < 1700, "Early Modern",
        IF(?year < 1900, "Modern",
    "Contemporary")
      )
    ) AS ?period
  )

  # Group countries into regions (simplified)
  BIND(
    IF(REGEX(?country,
    "France|Spain|Italy|Germany", "i"),
    "Western Europe",
      IF(REGEX(?country,
    "England|Scotland|Ireland", "i"),
    "British Isles",
        IF(REGEX(?country,
    "Russia|Poland|Hungary", "i"), "Eastern
    Europe",
          "Other"
        )
      )
    ) AS ?region
  )
}
GROUP BY ?period ?region
ORDER BY ?period ?region
```

Query 2. Cross-Cultural Comparison of Cipher Complexity

```
SELECT ?culture ?avgSymbolSetSize
    ?maxSymbolSetSize
WHERE {
  {
    SELECT ?culture (AVG(?symbolSetSize) AS
    ?avgSymbolSetSize) (MAX(?symbolSetSize)
    AS ?maxSymbolSetSize)
    WHERE {
      ?cipher decryptonto:hasCurrentCountry
    ?country ;
              decryptonto:hasSymbolSets
    ?symbolSets .

      # Simplified culture grouping
      BIND(
        IF(REGEX(?country,
    "China|Japan|Korea", "i"), "East Asian",
          IF(REGEX(?country,
    "Arabia|Persia|Ottoman", "i"), "Islamic",
            IF(REGEX(?country,
    "Italy|France|Spain", "i"), "Latin",
              IF(REGEX(?country,
    "England|Germany|Netherlands", "i"),
    "Germanic",
                "Other"
              )
            )
          )
        ) AS ?culture
      )

      # Assuming symbolSets is a
    comma-separated list, count the items
      BIND(1 + STRLEN(?symbolSets) -
    STRLEN(REPLACE(?symbolSets, ",", "")) AS
    ?symbolSetSize)
    }
    GROUP BY ?culture
  }
}
ORDER BY DESC(?avgSymbolSetSize)
```

	period	region	cipherCount
1	Contemporary	British Isles	"6"^^<http://www.w3.org/2001/XMLSchema#integer>
5	Early Modern	Eastern Europe	"114"^^<http://www.w3.org/2001/XMLSchema#integer>
10	Medieval	Western Europe	"699"^^<http://www.w3.org/2001/XMLSchema#integer>
11	Modern	British Isles	"670"^^<http://www.w3.org/2001/XMLSchema#integer>

Figure 10: A partial view of Query 1 results.

	culture	avgSymbolSetSize	maxSymbolSetSize
1	Latin	"1.819563780568407138136153"^^<http://www.w3.org/2001/XMLSchema#decimal>	"4"^^<http://www.w3.org/2001/XMLSchema#integer>
2	Other	"1.536078845476944737768391"^^<http://www.w3.org/2001/XMLSchema#decimal>	"4"^^<http://www.w3.org/2001/XMLSchema#integer>
3	Germanic	"1.264308681672025723472669"^^<http://www.w3.org/2001/XMLSchema#decimal>	"4"^^<http://www.w3.org/2001/XMLSchema#integer>

Figure 11: Results for Query 2.