Lagrange-DFA Offline Verification Scheme

Alex Jiang *

Department of Mathematics, University of California, Berkeley

June 2025

1 Introduction

This document describes an original cryptographic scheme for offline verification of whether an input string matches a pre-encoded secret pattern. This scheme is **not** intended to be a general-purpose cryptographic primitive. It is efficient to break in linear time, and any derivative, including the associated proof-of-concept work (see ZekaEngine at the end of the Discussion section), is efficient to break in at most polynomial time. Consequently, the construction should be understood only as a niche mechanism in the very specific problem setting for which it was designed, and it should be used only within appropriate operating parameters (see the Discussion section). This is not research associated with my institution. Rather, it is a semi-formal "idea dump" written independently whilst studying there, intended to capture and communicate a basic concept I thought of. If you find errors, which is highly likely, please email me.

While this document introduces ideas in the context of strings and regular expressions, it can easily be extended to any data or deterministic finite automata (DFA). This enables the possibility of encoding an arbitrary number of string-regex pairs in the same polynomial; the possibility of matching a string against an entire collection of regex, succeeding only if certain subsets of these regex validate; and so on, with increasing sophistication.

2 Notation

Let p be a large prime. Let d denote an input string to be verified against a regular expression R. Let $\mathbf{AES\_GCM}_k(m)$ denote AES encryption in CTR mode with key k, plaintext m, and a nonce IV, outputting a ciphertext-tag pair (c, t). Likewise, let $\mathbf{AES\_GCM}_k(c, t)$ denote AES decryption in CTR mode with key k, ciphertext c, authentication tag t, and a nonce IV, outputting

*Electronic address: jiang.alexander@berkeley.edu

plaintext m if successful. Let $\mathbf{HASH}(m)$ denote a hash function whose digest is appropriately sized for an AES-GCM key (i.e., SHA256 is appropriate for AES256-GCM).

Let $L_S : \mathbb{F}_p \to \mathbb{F}_p$ denote the Lagrange polynomial over $\mathbb{F}_p$, which interpolates a set of points $S \subseteq \mathbb{F}_p^2$ with minimal degree. Outside of the support points (i.e., the points in S), L_S still yields values, albeit unrelated to any point in S. Let M be the DFA representation of R and be defined by $\{(Q, \Sigma, \delta, q_0, F)\}$, where $Q \subseteq \mathbb{F}_p$ is the set of states, $\Sigma = \{0,1\}^8$ is the byte alphabet, $\delta : Q \times \Sigma \to Q$ is the transition function, $q_0 \in Q$ is the initial state, and $F \in Q$ is the singleton set containing the accepting state. Typically, the definition of F is more lax, but this scheme requires a singular accepting state. Let $\mathbf{PAIR}_p(a, b)$ denote an arbitrary pairing function modulo p.

3 Construction

For each valid transition in M, given by a state q, byte $c \in \Sigma$, and next state q', compute the point $(\mathbf{PAIR}_p(q, c), q')$. Collect these points into a set T and construct L_T by Lagrange interpolation. Then, given a plaintext m, compute $(c, t) = \mathbf{AES\_GCM}_{\mathbf{HASH}(F)}(m)$. Finally, make $L_T, c, t,$, the nonce IV, and q_0 public.

4 Usage

To verify an arbitrary string d, walk the DFA transitions:

Algorithm 1 Verifying a string

```
state ← q_0
for char ← d do
    state ← L_T(PAIR_p(state, char))
end for
```

Finally, attempt to compute $\mathbf{AES\_GCM}_{\mathbf{HASH}(state)}(c, t)$.

5 Discussion

To break the confidentiality of this scheme without knowing the accepting state F, an adversary's best generic strategy is to try candidate keys derived from every possible state in the image of L_T. As such, it is important to ensure that $|\mathrm{Im}(L_T)| > 2^{128}$ to ensure an appropriate security level. A lower bound for $|\mathrm{Im}(L_T)|$ can be determined by the argument that follows:

Let $f : \mathbb{F}_p \to \mathbb{F}_p$ be a polynomial of degree $d \geq 1$. For all $y \in \mathbb{F}_p$, the polynomial $f(x) - y$ is nonzero of degree d, thus has at most d solutions in $\mathbb{F}_p$. Thus, no fiber $f^{-1}(y)$ contains more than d elements. Since the fibers over $\mathrm{Im}(f)$ form a partition of $\mathbb{F}_p$, the sum of their cardinalities is simply p. All in all, we have:

$$p = \sum_{y \in \mathrm{Im}(f)} |f^{-1}(y)| \leq |\mathrm{Im}(f)| \cdot d$$

Rearranging yields:

$$|\mathrm{Im}(f)| \geq \frac{p}{d}$$

In other words, ensure p is substantially larger than d. In ZekaEngine, the bit length of p is at minimum 256. Since it is infeasible to interpolate roughly 2^{128} transitions, this p and d combination is appropriate in essentially every case. It is also important to note that the set of reachable states from any given starting state is not necessarily of the same cardinality as $\mathrm{Im}(f)$. The lower bound for this number of reachable states is trivial (consider the starting state 0; the set of reachable sets would then be singleton), and I am not mathematically equipped to continue analyzing this limitation or to present possible solutions.

Additionally, the use of a pairing function within a finite field is suspect. Any function $\mathbf{PAIR}_p : Q \times \Sigma \to \mathbb{F}_p$ must be injective to guarantee the uniqueness and correctness of transitions. However, if $|Q||\Sigma| > p$, then no injective mapping exists by the pigeonhole principle, so distinct input pairs will necessarily collide. Moreover, the underlying philosophy of the scheme is to treat Q as the entire field $\mathbb{F}_p$, because constructing a pairing function tailored specifically to the valid state set Q would itself expose the exact set of states to brute force. Thus, Q has cardinality p, and for any non-trivial Σ, $p \cdot |\Sigma| > p$. As such, any attempt to embed $Q \times \Sigma$ into $\mathbb{F}_p$ is inherently lossy, which makes the use of such a function fundamentally flawed in this context. A more appropriate solution is to use a multivariate interpolation polynomial, which I did not have enough time to research. This is something I would like to implement in the future.

The proof-of-concept project associated with this document, which extends this scheme extensively, is ZekaEngine, a cross-platform scoring engine I designed and implemented for "find-and-fix" cybersecurity exercises. In these competitions, the scoring engine monitors teams' systems and awards points when remediations are correctly implemented for existing vulnerabilities. If the scoring logic can be easily reversed or inferred, competitors can bypass the intended educational challenge. Likewise, if the engine lacks flexible configuration, it cannot handle the increasing complexity of modern competition environments. ZekaEngine is a partially good solution that addresses both of these shortcomings, but in general, a more appropriate solution consists of a matching algorithm over entirely *encrypted* DFAs. To my knowledge, this does not yet exist in the single-party context, which these exercises inherently demand. To design this is the long-term goal of ZekaEngine.