BotBrake: Privacy-Preserving Abuse Detection

Frank Denis

Abstract. Web services face an impossible choice: they need to detect bot attacks, but privacy regulations increasingly restrict access to the very data needed for detection. We built BotBrake to resolve this tension: it detects malicious traffic patterns without ever decrypting IP addresses or URLs.

The system works because bot behavior creates structural patterns that survive encryption. When a scanner probes hundreds of random paths, that randomness is still visible in the encrypted prefixes. When a botnet hammers login endpoints from a network block, the concentration pattern remains detectable. BotBrake exploits prefix-preserving encryption (IPCrypt-pfx for IPs, URICrypt for URIs) to maintain these hierarchical relationships while providing cryptographic privacy guarantees.

Rather than relying on a single detection method, BotBrake combines eight orthogonal signals (from Bayesian error modeling to robust statistical anomaly detection), each designed with what we call "false positive paranoia." Every algorithm includes multiple safeguards, uncertainty quantification, and conservative thresholds. The system would rather miss an attack than block a legitimate user.

The system is designed to identify various attack patterns: distributed credential stuffing, coordinated scanning from cloud providers, and network-level attacks where multiple IPs from the same block coordinate to stay below individual detection thresholds. All detection operates entirely on encrypted data, preserving privacy while maintaining security.

1 Introduction

Here's the fundamental problem: bots now generate between 40 and 60 percent of web traffic [5]. They're credential stuffing login endpoints, scraping content, buying up limited inventory, and launching increasingly sophisticated DDoS attacks. These attacks must be stopped. But privacy laws like GDPR treat IP addresses as personally identifiable information, and compliance teams are (rightfully) nervous about logging and analyzing user data.

The standard "solutions" don't actually solve anything. Truncating IP addresses to /24 blocks? That could be one user or an entire office building; there's no way to know. Hashing everything? Then it becomes impossible to detect that 50 IPs from the same network are coordinating an attack. Generic encryption makes the data completely opaque to analysis. Organizations are stuck choosing between user privacy and site security.

We built BotBrake to escape this trap. The system performs all detection on encrypted network data; it never sees the real IP addresses or URLs. It works because the patterns that reveal bot behavior aren't in the specific values but in their relationships. A scanner hitting random paths creates high entropy in the prefix structure. A botnet hammering login endpoints creates extreme concentration. These patterns survive encryption when hierarchical relationships are preserved.

The technical foundation relies on two cryptographic constructions. IPCrypt-pfx [1] encrypts IP addresses while preserving network prefixes: if two addresses are from the same /24 subnet, their encrypted forms will share a common prefix. This lets us detect coordinated attacks from network blocks without knowing which network. URICrypt does something similar for URIs using chained encryption, where each path component's encryption depends on all preceding components. When a scanner probes /admin/users, /admin/posts, and /admin/config, we see three different encrypted paths that all share the encrypted prefix for /admin/.

The detection system combines eight different signals because no single pattern catches all attacks. Here's what actually happens in practice:

– **Error patterns**: We use Beta-Binomial models because they handle uncertainty properly. A scanner making 10 requests with 10 errors looks suspicious, but the posterior probability is only

85% (not enough to block). The same scanner making 1000 requests with 1000 errors has posterior probability 99.9999% (clear abuse). The math automatically handles the "small sample problem" that plagues simpler approaches.

- **Exploration detection**: We compute robust z-scores using Median Absolute Deviation (MAD) instead of standard deviation. Why? Because web traffic has massive outliers, and one viral spike shouldn't corrupt our baseline. MAD has a 50% breakdown point: half the data can be corrupted and it still works.
- **Traffic concentration**: This catches the opposite pattern: instead of exploring everywhere, the attacker hammers one endpoint. But here's the trick: for dominant networks ($> 30\%$ of traffic), we check total concentration across all paths, not just the top path. This catches distributed attacks that spread across multiple URLs to evade detection.
- **Rate anomalies**: EWMA tracks the adaptive baseline while CUSUM detects sustained shifts. They're complementary: EWMA catches sudden spikes, CUSUM catches gradual escalation that EWMA would adapt to.
- **Persistence tracking**: Attacks run continuously; legitimate traffic is bursty. If signals trigger in 3+ consecutive windows, that adds 20-60 extra points to the score.

Every algorithm embodies what we call "false positive paranoia." We'd rather let some attacks through than block legitimate users. This philosophy drives every design decision: conservative statistical thresholds, multi-signal consensus requirements, dampeners for low-volume entities, and special protection for new content that might cause legitimate traffic spikes.

We present the following mechanisms:

1. **Privacy-preserving detection framework**: A comprehensive system for bot detection operating entirely on encrypted network data, providing privacy guarantees while maintaining operational effectiveness.
2. **URICrypt specification**: A prefix-preserving encryption scheme for URIs using chained TurboSHAKE128 operations, enabling structural analysis of encrypted web paths with authenticated encryption properties.
3. **Multi-signal consensus architecture**: A robust detection framework combining eight orthogonal signals with principled statistical foundations, consensus requirements, and synergy bonuses for known attack patterns.

2 IPCrypt: Prefix-Preserving IP Address Encryption

2.1 Motivation and Requirements

IP addresses serve as fundamental identifiers in network traffic, but their exposure creates significant privacy risks. A single IP address can reveal geographic location, organization affiliation, and when correlated across time, detailed behavior patterns. However, complete encryption destroys the network structure essential for security analysis. Addresses from the same subnet appear unrelated after standard encryption, preventing detection of coordinated attacks from network blocks.

Prefix-preserving encryption resolves this tension by maintaining network relationships in encrypted form. Addresses sharing a network prefix (e.g., 192.168.1.0/24) produce ciphertexts sharing an encrypted prefix. This enables network-level analysis, including detecting scanning from a subnet, identifying distributed attacks, and recognizing organizational traffic patterns, while preventing identification of actual networks.

2.2 Cryptographic Construction

IPCrypt-pfx implements prefix-preserving encryption through bit-by-bit transformation using a pseudorandom function (PRF). For each bit position i from most significant to least significant:

1. Extract the prefix $p_{0..i-1}$ containing all bits processed so far

2. Compute a PRF output using the padded prefix as input
3. XOR the least significant bit of PRF output with the original bit b_i
4. The resulting bit depends deterministically on the prefix, ensuring consistency

The PRF construction uses two independently keyed AES-128 encryptions:

$$\mathrm{PRF}_K(x) = \mathrm{AES}_{K_1}(x) \oplus \mathrm{AES}_{K_2}(x) \tag{1}$$

where the 256-bit key K is split into two 128-bit keys K_1 and K_2. This XOR construction provides provable security: the sum of two independent pseudorandom permutations is a secure PRF up to 2^{128} queries [3, 4].

2.3 Padding and Domain Separation

For each bit position, the current prefix must be padded to 128 bits for AES input. The padding format ensures unique inputs across iterations:

$$\text{padded\_prefix} = \underbrace{0...0}_{\text{zeros}} \parallel 1 \parallel \underbrace{p_0 p_1 ... p_{i-1}}_{\text{prefix bits}} \tag{2}$$

The single '1' bit serves as a delimiter, preventing ambiguity between prefixes of different lengths. This construction provides domain separation: each bit position receives a unique padded input even when processing the same prefix values.

2.4 Algorithm Specification

Algorithm 1 IPCrypt-pfx Encryption

Require: IP address $addr$, 256-bit key K
Ensure: Encrypted IP address enc

$K_1 \leftarrow K[0:127]$, $K_2 \leftarrow K[128:255]$
$enc \leftarrow \texttt{zeros}(|addr|)$ {Maintain native size}
padded_prefix $\leftarrow$ pad_prefix($\emptyset$)
for $i = 0$ to $|addr| - 1$ **do**
  $e_1 \leftarrow \text{AES-128}_{K_1}(\text{padded\_prefix})$
  $e_2 \leftarrow \text{AES-128}_{K_2}(\text{padded\_prefix})$
  $e \leftarrow e_1 \oplus e_2$
  cipher_bit $\leftarrow e[0]$ {LSB of PRF output}
  $enc[i] \leftarrow addr[i] \oplus$ cipher_bit
  padded_prefix $\leftarrow$ shift_left(padded_prefix, 1)
  padded_prefix[0] $\leftarrow addr[i]$
end for
return enc

2.5 Security Properties

Theorem 1. *The* IPCrypt-*pfx construction is a secure prefix-preserving encryption scheme with distinguishing advantage bounded by* $q^2/2^{129}$ *for* q *queries, assuming AES is a pseudorandom permutation.*

Proof sketch. The security follows from the PRF-security of the XOR construction. Each bit position uses a unique padded input, preventing related-key attacks. The XOR of two independent PRP outputs is indistinguishable from random up to the birthday bound [4].

2.6 Performance Characteristics

IPCrypt-pfx requires $2n$ AES operations for an n-bit address:

- IPv4 (32 bits): 64 AES operations
- IPv6 (128 bits): 256 AES operations

While computationally intensive compared to single-block encryption, the construction is highly parallelizable. Modern processors with AES-NI instructions achieve sub-microsecond encryption times. Caching encrypted prefixes for common subnets provides significant optimization for traffic with locality.

3 URICrypt: Hierarchical URI Encryption

3.1 Design Goals

Web URIs encode hierarchical resource structures through path components separated by delimiters. Standard encryption destroys this structure, preventing analysis of traversal patterns essential for detecting scanning attacks. URICrypt provides prefix-preserving encryption for URIs while adding authenticated encryption properties absent from IPCrypt.

Key requirements:

- **Prefix preservation**: URIs sharing path prefixes produce ciphertexts with shared encrypted prefixes
- **Component integrity**: Prevent tampering, reordering, or mixing of path segments
- **Variable length support**: Handle arbitrary component lengths without revealing exact sizes
- **Efficient encoding**: Produce compact, URL-safe output

3.2 Cryptographic Architecture

URICrypt employs chained encryption where each component's encryption depends on all preceding components. This creates a cryptographic chain ensuring prefix preservation while providing authentication:

1. Initialize an extendable-output function (XOF) with the secret key
2. For each path component in sequence:
   - Update XOF state with the plaintext component
   - Generate a Synthetic Initialization Vector (SIV) from accumulated state
   - Derive component-specific keystream using SIV
   - Encrypt component via XOR with keystream
   - Output SIV concatenated with encrypted component

The chained construction ensures identical prefixes always produce identical encrypted prefixes, while the SIV provides authentication detecting any modification.

3.3 XOF-Based Construction

URICrypt uses TurboSHAKE128 [2] as its cryptographic primitive. The XOF initialization incorporates domain separation:

The dual XOF design separates SIV generation from keystream production, preventing related-key vulnerabilities while maintaining efficiency through state cloning.

Algorithm 2 URICrypt XOF Initialization

```
base_xof ← TurboSHAKE128(0x1F)
base_xof.update(len(key) ‖ key)
base_xof.update(len(context) ‖ context)
components_xof ← base_xof.clone()
components_xof.update("IV")
keystream_xof ← base_xof.clone()
keystream_xof.update("KS")
```

3.4 Component Processing

For each URI component, encryption proceeds as:

$$\begin{aligned} \text{components\_xof} &\leftarrow \text{update}(\text{component}) \\ \text{SIV} &\leftarrow \text{squeeze}(\text{components\_xof}, 128) \\ \text{keystream\_xof} &\leftarrow \text{clone}(\text{base\_keystream\_xof}) \\ \text{keystream\_xof} &\leftarrow \text{update}(\text{SIV}) \\ \text{keystream} &\leftarrow \text{squeeze}(\text{keystream\_xof}, |\text{component}|) \\ \text{ciphertext} &\leftarrow \text{component} \oplus \text{keystream} \\ \text{output} &\leftarrow \text{SIV} \parallel \text{ciphertext} \end{aligned} \tag{3}$$

The SIV serves dual purposes: domain separation for keystream generation and authentication tag for integrity verification.

3.5 Padding and Encoding

To produce clean base64url output without padding characters, each encrypted component is padded to a multiple of 3 bytes:

$$\text{padding\_len} = (3 - (|\text{SIV}| + |\text{component}|) \bmod 3) \bmod 3 \tag{4}$$

This ensures base64 encoding produces aligned output while providing partial length hiding. Components differing by 1-2 bytes may produce identical ciphertext lengths.

3.6 Security Analysis

Theorem 2. URICrypt *provides semantic security for components and existential unforgeability with advantage bounded by* $q/2^{128}$ *for* q *queries, assuming TurboSHAKE128 is a secure XOF.*

Proof sketch. Each component receives a unique keystream derived from its SIV. The SIV depends on all previous components through the chained XOF state, preventing reordering attacks. Forgery requires finding a collision in the 128-bit SIV space, occurring with probability $q/2^{128}$ for q attempts.

3.7 Example Encryption

Consider the URI path `/api/v1/users/profile`:

1. Component 1: `api/`
   - Update XOF with "api/"
   - Generate SIV_1 (16 bytes)
   - Encrypt "api/" $\rightarrow \text{SIV}_1 \parallel \text{enc}_1$
2. Component 2: `v1/`
   - Update XOF with "v1/" (state includes "api/")

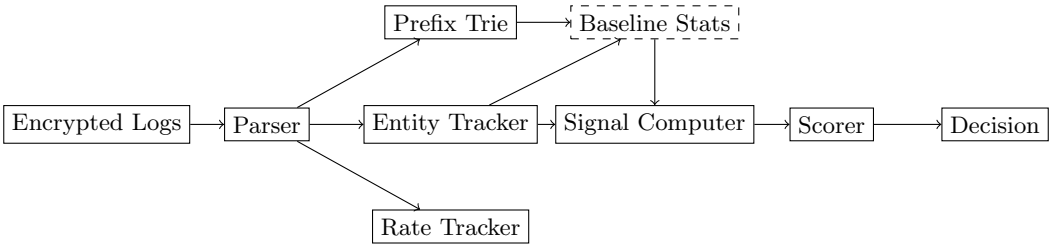

Fig. 1. BotBrake data flow architecture. Encrypted logs flow through parsing, entity tracking, signal computation, and scoring to produce blocking decisions.

- Generate SIV_2 (depends on "api/")
- Encrypt "v1/" $\rightarrow \text{SIV}_2 \parallel \text{enc}_2$

3. Components 3-4: Similar process
4. Final output: Base64url($\text{SIV}_1 \parallel \text{enc}_1 \parallel \text{SIV}_2 \parallel \text{enc}_2 \parallel ...$)

URIs sharing the prefix `/api/v1/` will have identical SIV_1, enc_1, SIV_2, enc_2 values, preserving structure while encrypting content.

4 System Architecture

4.1 Overview

BotBrake implements a two-pass architecture: training followed by detection. The training phase processes the first hour of traffic to establish baseline statistics. The detection phase analyzes subsequent traffic in sliding windows, comparing observed patterns against learned baselines to identify abuse.

4.2 Entity Model

BotBrake tracks four entity types, each offering different perspectives on traffic:

- **IP addresses**: Individual encrypted addresses
- **User Agents**: Browser/client identifiers
- **CIDR blocks**: Network prefixes (/24 IPv4, /48 IPv6)
- **URI paths**: Encrypted resource paths

Per-entity metrics accumulate within time windows:

- Request counts and error rates
- Unique path cardinality
- Path frequency distributions
- Status code breakdowns
- Temporal patterns

4.3 Sliding Window Management

Three window sizes capture different attack timescales:

- **60 seconds**: Fast attacks, burst behavior
- **300 seconds**: Sustained campaigns
- **3600 seconds**: Persistent, low-rate abuse

Windows slide continuously rather than tumbling, ensuring attacks spanning boundaries are captured. Each window maintains independent entity metrics, enabling multi-timescale analysis.

4.4 Prefix Trie for URICrypt Analysis

Encrypted URI paths are stored in a prefix trie structure enabling efficient prefix analysis. Each node tracks:

- Hit counts per prefix depth
- Unique IP/UA sets
- Time first seen (TFS)
- Error rate per prefix

The trie enables BotBrake to compute prefix fanout, which counts how many unique prefixes exist at each depth. This metric is crucial for distinguishing focused legitimate traffic from scanning attempts.

4.5 Two-Pass Training Pipeline

Pass 1: Baseline Learning The first hour of traffic establishes normal patterns:

Algorithm 3 Training Phase

```
for each record in first hour do
    Update entity metrics
    Update prefix trie
    if entity appears legitimate then
        Collect error rate samples
        Record exploration ratios
        Track depth distributions
    end if
end for
Fit Beta-Binomial prior from error rates
Compute MAD for each metric
Initialize EWMA baselines
```

The system learns the following key statistics:

- **Error rate prior**: Beta distribution parameters (α, β) via method of moments
- **Exploration baseline**: Median and MAD of unique paths to total requests ratios
- **Depth distribution**: Typical navigation depths in the site structure
- **Rate baselines**: EWMA initialization for traffic rates

Pass 2: Detection Detection compares real-time metrics against baselines:

4.6 CIDR Aggregation

Before individual IP analysis, BotBrake aggregates metrics by CIDR blocks to detect distributed attacks:

$$\text{CIDR}_{/24}(\text{IP}) = \text{IP}[0:23] \parallel \text{zeros}(8) \tag{5}$$

Aggregated metrics reveal network-level patterns invisible in per-IP analysis. A botnet distributing requests across 50 IPs might keep each below thresholds, but the aggregate CIDR metrics expose coordination.

Algorithm 4 Detection Phase

```
for each time window do
    for each entity in window do
        Compute 8 signal components
        Apply dampeners (volume, new content)
        Calculate weighted score
        if score ¿ threshold AND consensus met then
            Map score to block duration
            Emit decision
        end if
    end for
end for
```

5 Multi-Signal Detection Algorithms

5.1 S1: Error Propensity (Beta-Binomial Model)

Here's a practical problem: consider an IP making 5 requests with 5 errors. Is that a scanner probing for vulnerabilities, or just someone who hit a broken link? The naive approach (flag anyone above 50% error rate) would block them immediately. But 5 requests isn't enough evidence.

This is why we use Beta-Binomial modeling. It's the mathematically correct way to handle proportions with small samples. The prior Beta(α_0, β_0) represents our baseline belief about error rates learned from legitimate traffic. For a typical site, that's around 10% errors. When we see x errors in n requests, we update to the posterior:

$$\text{Posterior} = \text{Beta}(\alpha_0 + x, \beta_0 + n - x) \tag{6}$$

What happens to our 5/5 errors case? With a Beta(2, 18) prior (10% baseline), the posterior mean is only 0.38, heavily shrunk from the empirical 1.0. The model is saying: "Yes, 100% errors looks bad, but with only 5 requests, it's probably just bad luck." The posterior probability this entity exceeds 1.5× baseline is about 85%, suspicious but not conclusive.

Now consider 500/500 errors. The posterior mean jumps to 0.98, and the probability of exceeding 1.5× baseline is 99.9999%. That's definitive abuse. The model automatically adjusts its confidence based on sample size:

$$s_{\text{err}} = 100 \times P(\theta > 1.5 \times \theta_{\text{baseline}}) \tag{7}$$

Why not use a t-test or simple threshold? Because they don't handle the fundamental uncertainty properly. A frequentist t-test assumes normality (wrong for binomial data) and treats 5/5 the same as 500/500 in terms of the point estimate. The Bayesian approach captures exactly what we need: more data means more confidence.

5.2 S2: Exploration Behavior (Scanning Detection)

Scanners have a distinctive pattern: they explore everywhere. A legitimate user might visit 10 pages in 100 requests; they find what they want and stick around. A scanner hits 100 different paths in 100 requests; they're mapping the entire site structure.

We detect this through multiple lenses. The simplest is the exploration ratio:

$$\text{explore\_ratio} = \frac{|\text{unique\_paths}|}{|\text{total\_requests}|} \tag{8}$$

But here's where it gets interesting. Standard deviation is useless for anomaly detection in web traffic because the data is full of outliers. One viral post can spike traffic 100x. With standard deviation, that spike becomes the new "normal" and real anomalies become invisible.

Instead, we use Median Absolute Deviation (MAD) for robust z-scores:

$$\text{robust\_z} = \frac{\text{value} - \text{median}}{1.4826 \times \text{MAD}} \tag{9}$$

Why MAD? It has a 50% breakdown point. Half the data could be corrupted with outliers and MAD would still provide the right baseline. Standard deviation has a 0% breakdown point: a single outlier ruins everything. The 1.4826 constant makes MAD comparable to standard deviation for normal data, so our z-score thresholds still make intuitive sense.

But exploration ratio alone isn't enough. Sophisticated scanners try to blend in. So we also check:

- **Prefix fanout at multiple depths**: A scanner might probe /admin/users, /admin/posts, /admin/config. Low exploration ratio (all under /admin/) but high fanout at depth 6 reveals the scanning pattern.
- **Depth score**: Legitimate users navigate deep into sites (articles, profiles, nested content). Scanners stay shallow; they're just checking if paths exist.

We take the maximum z-score across all these metrics and transform it:

$$s_{\text{explore}} = \frac{100}{1 + e^{-0.5(z_{\max} - 4)}} \tag{10}$$

The logistic function gives us a smooth score from 0-100. At z=4 (our detection threshold), the score is exactly 50. Below z=2, the score is near zero. Above z=6, it saturates near 100. This prevents a single extreme outlier from dominating the final score.

5.3 S3: Traffic Concentration (Hammering Detection)

Hammering is the opposite of scanning: instead of exploring everywhere, the attacker focuses on one or a few endpoints. Think credential stuffing on /login, or a DDoS targeting expensive search endpoints. Simple, right? Just check if most requests go to the same path.

But there's an interesting problem with coordinated attacks: multiple IPs from the same network block might distribute their requests across a small set of URLs. Each URL individually might receive only 25% of the traffic (below our single-path threshold), but 99%+ of the network's traffic could concentrate on just those few URLs. That's not normal browsing behavior.

This led us to a two-tier detection system. For normal entities:

$$s_{\text{hammer}} = 100 \times \text{clamp}\left(\frac{\text{top\_path\_ratio} - 0.5}{0.4}, 0, 1\right) \tag{11}$$

If more than 50% of requests hit the same path, hammering signal starts accumulating. At 90%, the score maxes out at 100 points.

But for dominant entities (generating $> 30\%$ of total traffic), we switch strategies. Instead of looking at single-path concentration, we measure total concentration:

$$\text{concentration} = 1 - \text{explore\_ratio} \tag{12}$$

If 99%+ of traffic focuses on just a handful of paths (even if no single path dominates), that's coordinated hammering:

$$s_{\text{hammer}} = 100 \times \frac{\text{concentration} - 0.99}{0.01} \quad \text{if concentration} \geq 0.99 \tag{13}$$

Why 99%? Because legitimate traffic, even from a CDN or corporate network, shows more diversity. Users browse around. They click links. They explore. Traffic that's 99% concentrated on a few paths is either a misconfigured system or an attack. Either way, it needs to stop.

We also require minimum volume (500 requests) before computing hammering signals. Otherwise, a user refreshing their homepage 5 times would trigger a false positive.

5.4 S4: Network Dominance

Networks generating disproportionate traffic often indicate coordinated abuse:

$$s_{\text{dominance}} = \begin{cases} 100 \times \text{clamp}\left(\frac{\text{traffic\_ratio}-0.3}{0.3}, 0, 1\right) & \text{if ratio} \geq 0.3 \\ 0 & \text{otherwise} \end{cases} \tag{14}$$

The 30% threshold balances detection of abuse with tolerance for legitimate concentrated sources (CDNs, corporate networks).

5.5 S5: Rate Anomalies (EWMA and CUSUM)

Rate spikes indicate attacks but legitimate traffic also varies. BotBrake uses complementary approaches:

EWMA for adaptive baseline:

$$\lambda_t = \alpha \times \text{rate}_t + (1 - \alpha) \times \lambda_{t-1}, \quad \alpha = 0.3 \tag{15}$$

Tail probability under Poisson model:

$$p_{\text{tail}} = P(\text{rate} \geq \text{current} | \lambda) \tag{16}$$

CUSUM for persistent shifts:

$$S_t = \max(0, S_{t-1} + \text{rate}_t - \mu - k), \quad k = 2\sigma \tag{17}$$

Signal combines both:

$$s_{\text{burst}} = 100 \times (1 - p_{\text{tail}})^{0.5} \tag{18}$$

The square root dampening prevents rate signals from dominating.

5.6 S6: Temporal Persistence

Sustained malicious behavior across windows confirms attacks:

$$s_{\text{persist}} = \min(100, 20 \times \text{consecutive\_windows\_with\_signals}) \tag{19}$$

Each consecutive window with signals adds 20 points, capping at 100 (5 windows).

5.7 S7: Entity Spread

Coordinated attacks manifest as unusual entity distributions:

For User Agents seen from many IPs:

$$s_{\text{spread}} = \min\left(100, \frac{|\text{unique\_IPs}| - 200}{5}\right) \tag{20}$$

For IPs using many User Agents (rotation):

$$s_{\text{spread}} = \min(100, |\text{unique\_UAs}| \times 10) \tag{21}$$

5.8 S8: Cross-Entity Correlation

When multiple entity types (IP, UA, path) show coordinated patterns, confidence increases:

$$s_{\text{cross}} = \min(100, 25 \times |\text{entities\_flagged}|) \tag{22}$$

This signal is currently a simplified implementation, with room for sophisticated correlation analysis.

6 Scoring and Decision Framework

6.1 Weighted Linear Combination

After computing all eight signals, we need a single score that determines whether to block. We use a weighted linear combination: simple, interpretable, and effective:

$$\text{Score}_{\text{base}} = \sum_{i=1}^{8} w_i \times s_i \tag{23}$$

The weights weren't chosen arbitrarily. We tuned them empirically on real attack traffic, but each weight has a clear justification:

Signal	Weight
Error (w_1)	0.28
Explore (w_2)	0.18
Hammer (w_3)	0.18
Dominance (w_4)	0.06
Burst (w_5)	0.12
Persist (w_6)	0.10
Spread (w_7)	0.05
Cross (w_8)	0.03

Errors get the highest weight (0.28) because they're the most reliable abuse indicator. Legitimate traffic rarely exceeds 10-15% error rates; scanners routinely hit 50-100%. Exploration and hammering get equal weight (0.18 each) because they represent two sides of the abuse spectrum: high diversity scanning and low diversity hammering. Together, these three primary signals account for 64% of the score.

The supporting signals have lower weights. Rate bursts (0.12) can indicate attacks but also legitimate viral traffic, so they need corroboration. Persistence (0.10) acts as a multiplier; it doesn't detect attacks itself but confirms sustained abuse. Dominance (0.06) and spread (0.05) are powerful when combined with other signals but rare on their own.

6.2 Dampeners

Three dampeners reduce false positives:

Volume dampener penalizes small samples:

$$d_{\text{volume}} = \begin{cases} 40 \times \left(1 - \frac{n}{n_{\min}}\right) & \text{if } n < n_{\min} \\ 0 & \text{otherwise} \end{cases} \tag{24}$$

New content dampener protects viral content:

$$d_{\text{new}} = 30 \times \frac{\text{requests\_to\_new\_content}}{\text{total\_requests}} \tag{25}$$

New content defined as: age < 90 minutes, 100+ unique IPs, error rate $< 20\%$.

Legitimate UA dampener for known good crawlers:

$$d_{\text{legit}} = 50 \times \mathbb{1}[\text{verified\_crawler}] \tag{26}$$

6.3 Synergy Bonuses

Specific attack patterns receive bonuses:

Redirect abuse (hammering + errors):

$$\text{If } s_{\text{hammer}} > 80 \wedge s_{\text{err}} > 40 : \text{Score+} = 37 \tag{27}$$

Network abuse (dominance + hammering):

$$\text{If } s_{\text{dominance}} > 35 \wedge s_{\text{hammer}} > 25 : \text{Score+} = 40 \tag{28}$$

Final score:

$$\text{Score} = \text{clamp}(\text{Score}_{\text{base}} - \sum d_j + \text{Synergies}, 0, 100) \tag{29}$$

6.4 Consensus Requirements

Here's a critical design decision: we require multiple independent signals before blocking an IP or User-Agent. Why? Because any single anomaly might have a legitimate explanation. High errors? Maybe they hit a broken link. High exploration? Could be a search engine crawler. But when high errors AND high exploration occur together? That's a scanner probing for vulnerabilities.

The consensus rules are straightforward:

- **IP/UA blocks**: Need at least 2 signals above 20 points each. This dramatically reduces false positives: if each signal has a 1% false positive rate, requiring two signals drops it to 0.01%.
- **Path blocks**: Only need 1 signal. Blocking a path is much safer than blocking a user; if we're wrong about a path being malicious, we just block access to potentially broken content.
- **CIDR blocks**: Normally require consensus, but we make an exception for the specific pattern of network dominance + hammering, which unambiguously indicates coordinated abuse.

We also bypass consensus for two specific attack patterns we see frequently:

- **DDoS pattern**: When both hammering and burst signals exceed 60 points, that's a classic DDoS: high volume concentrated on a single endpoint. No legitimate traffic looks like this.
- **Network abuse**: When a network dominates traffic (> 35 points) while showing low path diversity (> 25 hammering points), that's coordinated abuse. This pattern emerges when individual IPs look normal, but the network-level behavior is unmistakably malicious.

This consensus approach embodies our "false positive paranoia." We'd rather let marginal attacks through than accidentally block legitimate users.

6.5 Duration Mapping

Block duration scales exponentially with confidence:

$$\text{Duration} = \begin{cases} 15 + 3(\text{score} - 60) \text{ min} & 60 \leq \text{score} < 75 \\ 2^{(\text{score}-70)/10} \times 10 \text{ min} & 75 \leq \text{score} < 90 \\ 2^{(\text{score}-80)/7} \times 30 \text{ min} & \text{score} \geq 90 \end{cases} \tag{30}$$

Entity-specific thresholds:

- Path: 60 (lowest risk)
- IP/UA: 75 (standard)
- CIDR: 50 (specific pattern required)

7 Discussion

7.1 Privacy Guarantees and Limitations

BotBrake provides cryptographic privacy for individual addresses and paths. Without the encryption key, the original values cannot be recovered. However, the system explicitly preserves certain structural properties:

- **Network relationships**: Addresses from the same subnet produce related ciphertexts
- **Path hierarchies**: URI prefixes remain correlated in encrypted form
- **Temporal patterns**: Request timing and volumes remain visible
- **Error patterns**: HTTP status codes are not encrypted

These properties enable detection but also limit privacy. An adversary observing encrypted traffic can infer:

- Whether requests originate from the same network
- Relative popularity of different URI paths
- Timing correlations between requests
- Success/failure patterns

Organizations requiring stronger privacy guarantees might consider:

- Adding noise to timing (impacts real-time detection)
- Batching and shuffling requests (increases latency)
- Encrypting status codes (reduces detection accuracy)

7.2 Adversarial Considerations

Sophisticated attackers might attempt to evade detection through:

Mimicking legitimate patterns: Adversaries could analyze baseline statistics and craft traffic matching normal distributions. Defenses include:

- Multi-signal consensus makes complete mimicry difficult
- Behavioral signals (mouse movement, JavaScript execution) could supplement network analysis
- Periodic retraining on verified clean traffic

Distributed attacks: Spreading malicious traffic across many IPs and networks to stay below thresholds. Current defenses:

- CIDR aggregation detects network-level coordination
- Cross-entity correlation identifies distributed campaigns
- Lower thresholds for valuable resources (login endpoints)

Poisoning training data: Injecting malicious traffic during the training phase. Mitigations:

- Robust statistics (MAD) resist outlier poisoning
- Manual review of training baselines
- Gradual baseline updates with decay

Parameter Tuning While default parameters work well across diverse traffic, site-specific tuning improves accuracy:

- **Error thresholds**: APIs may have different error patterns than content sites
- **Window sizes**: Adjust based on traffic volume and attack patterns
- **Signal weights**: Emphasize signals matching common attacks
- **Consensus requirements**: Balance false positives vs. false negatives

8 Conclusion

BotBrake demonstrates that privacy and security aren't mutually exclusive. With the right cryptographic tools and statistical techniques, both goals can be achieved.

The approach extends beyond bot detection. Any security system that analyzes network patterns could adopt these techniques: DDoS mitigation, intrusion detection, fraud prevention. As privacy laws get stricter and attacks grow more sophisticated, this isn't just an academic exercise; it's the future of security infrastructure.

References

1. F. Denis. IPCrypt: Optimal, Practical Encryption of IP Addresses for Privacy and Measurement. *Cryptology ePrint Archive, Paper 2025/1689*, 2025. `https://eprint.iacr.org/2025/1689`.
2. G. Bertoni, J. Daemen, M. Peeters, G. Van Assche, and R. Van Keer. TurboSHAKE. *IACR Transactions on Symmetric Cryptology*, 2023.
3. S. Lucks. The Sum of PRPs Is a Secure PRF. In *EUROCRYPT 2000*, LNCS 1807, pages 470–484, 2000.
4. J. Patarin. A Proof of Security in $O(2^n)$ for the XOR of Two Random Permutations. In *ICITS 2008*, LNCS 5155, pages 232–248, 2008.
5. Imperva. 2023 Imperva Bad Bot Report. Technical report, Imperva, 2023.
6. P. Tan and J. Li. A Comprehensive Survey of Bot Detection Techniques. *ACM Computing Surveys*, 45(4), 2013.
7. Z. Chu, S. Gianvecchio, H. Wang, and S. Jajodia. Detecting Automation of Twitter Accounts: Are You a Human, Bot, or Cyborg? *IEEE Transactions on Dependable and Secure Computing*, 9(6):811–824, 2012.
8. F. Soldo, A. Le, and A. Markopoulou. Predictive Blacklisting as an Implicit Recommendation System. In *IEEE INFOCOM*, 2011.
9. F. Haddadi and A. N. Zincir-Heywood. Analyzing Network Traffic for Botnet Detection. *Journal of Network and Systems Management*, 22(4):543–566, 2014.
10. X. Yuan, C. Li, and X. Li. DeepDefense: Identifying DDoS Attack via Deep Learning. In *IEEE SMARTCOMP*, 2017.
11. L. von Ahn, M. Blum, N. Hopper, and J. Langford. CAPTCHA: Using Hard AI Problems for Security. In *EUROCRYPT 2003*, 2003.
12. M. Jakobsson and A. Juels. Proofs of Work and Bread Pudding Protocols. In *Communications and Multimedia Security*, 1999.
13. S. Sivakorn, J. Polakis, and A. D. Keromytis. I'm Not a Human: Breaking the Google reCAPTCHA. In *Black Hat Asia*, 2016.
14. L. Sweeney. k-Anonymity: A Model for Protecting Privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 10(5):557–570, 2002.
15. C. Dwork. Differential Privacy. In *ICALP 2006*, pages 1–12, 2006.
16. J. Xu, J. Fan, M. Ammar, and S. B. Moon. On the Design and Performance of Prefix-Preserving IP Traffic Trace Anonymization. In *ACM SIGCOMM IMW*, 2001.
17. A. C. Yao. Protocols for Secure Computations. In *FOCS*, pages 160–164, 1982.
18. C. Gentry. Fully Homomorphic Encryption Using Ideal Lattices. In *STOC*, pages 169–178, 2009.
19. V. Costan and S. Devadas. Intel SGX Explained. *IACR Cryptology ePrint Archive*, 2016.
20. W. Yurcik, C. Woolam, G. Hellings, L. Khan, and B. Thuraisingham. Scrub-tcpdump: A Multi-Level Packet Anonymizer Demonstrating Privacy/Analysis Tradeoffs. In *IEEE Security and Privacy Workshops*, 2007.
21. R. Pang, M. Allman, V. Paxson, and J. Lee. The Devil and Packet Trace Anonymization. *ACM SIGCOMM Computer Communication Review*, 36(1):29–38, 2006.
22. M. Bellare, T. Ristenpart, P. Rogaway, and T. Stegers. Format-Preserving Encryption. In *SAC 2009*, pages 295–312, 2009.

23. M. Bellare, P. Rogaway, and T. Spies. The FFX Mode of Operation for Format-Preserving Encryption. NIST submission, 2010.
24. V. Paxson and S. Floyd. Wide-Area Traffic: The Failure of Poisson Modeling. *IEEE/ACM Transactions on Networking*, 3(3):226–244, 1995.
25. W. E. Leland, M. S. Taqqu, W. Willinger, and D. V. Wilson. On the Self-Similar Nature of Ethernet Traffic. *IEEE/ACM Transactions on Networking*, 2(1):1–15, 1994.
26. P. J. Rousseeuw and A. M. Leroy. *Robust Regression and Outlier Detection*. Wiley, 1987.
27. A. Gelman, J. B. Carlin, H. S. Stern, D. B. Dunson, A. Vehtari, and D. B. Rubin. *Bayesian Data Analysis*. CRC Press, third edition, 2013.
28. B. Efron. Large-Scale Inference: Empirical Bayes Methods for Estimation, Testing, and Prediction. Cambridge University Press, 2012.
29. E. S. Page. Continuous Inspection Schemes. *Biometrika*, 41(1/2):100–115, 1954.
30. S. W. Roberts. Control Chart Tests Based on Geometric Moving Averages. *Technometrics*, 1(3):239–250, 1959.
31. J. Kleinberg. Bursty and Hierarchical Structure in Streams. *Data Mining and Knowledge Discovery*, 7(4):373–397, 2003.
32. R. Sommer and V. Paxson. Outside the Closed World: On Using Machine Learning for Network Intrusion Detection. In *IEEE Symposium on Security and Privacy*, 2010.
33. M. Burkhart, M. Strasser, D. Many, and X. Dimitropoulos. SEPIA: Privacy-Preserving Aggregation of Multi-Domain Network Events and Statistics. In *USENIX Security Symposium*, 2010.
34. S. T. Zargar, J. Joshi, and D. Tipper. A Survey of Defense Mechanisms Against Distributed Denial of Service (DDoS) Flooding Attacks. *IEEE Communications Surveys & Tutorials*, 15(4):2046–2069, 2013.
35. M. Green and I. Miers. Forward Secure Asynchronous Messaging from Puncturable Encryption. In *IEEE Symposium on Security and Privacy*, 2015.
36. A. Boldyreva, N. Chenette, Y. Lee, and A. O'Neill. Order-Preserving Symmetric Encryption. In *EUROCRYPT 2009*, 2009.
37. D. Boneh, A. Sahai, and B. Waters. Functional Encryption: Definitions and Challenges. In *TCC 2011*, 2011.