**Crypto Forensics: Overview, Open-Source Tools, and Implementation Guide**

**1) What is Crypto Forensics?**

**Crypto (blockchain) forensics** is the practice of collecting, analyzing, linking, and interpreting blockchain and related digital evidence to answer investigative questions: who controlled an address, where funds moved, whether transactions are linked to illicit activity, or to produce audit trails for compliance. It combines:

* **On-chain analysis** (blocks, transactions, addresses);
* **Off-chain enrichment** (exchange API data, KYC lists, sanction lists, darknet indicators, OSINT);
* **Entity resolution** (cluster addresses that likely belong to the same actor);
* **Network/graph analytics** (flow paths, clustering, centrality, time-series flow);
* **Endpoint and memory forensics** (recover wallets, keys, private data from seized devices);
* **Reporting & chain-of-custody** for legal/forensic use.

Crypto forensics is both investigative (fraud, money laundering, theft) and preventive (compliance monitoring, suspicious activity detection).

**2) Notable Free & Open-Source Tools (with short descriptions)**

These are practical, community-backed projects you can use or extend.

**Blockchain data ingestion / ETL**

* **blockchain-etl / ethereum-etl** — tools to export blockchain data (blocks, transactions, token transfers) to CSV/BigQuery/Parquet. Useful to build analytics pipelines.
* **Bitcoin Core** — reference Bitcoin full node. Essential for authoritative on-chain data.
* **Geth** / **Erigon** / **OpenEthereum** — Ethereum clients for reading chain data.

**Blockchain analytics / graph tools**

* **BlockSci** — an academic-grade Bitcoin blockchain analysis framework (C++ core + Python bindings) for fast address clustering and graph queries.
* **GraphSense** — open-source blockchain analytics platform focused on entity clustering and analyst workflows.
* **OXT (OXT.me codebase / Samourai OXT)** — open Bitcoin analytics / explorer resources (OXT provides insights & research).

**General data & graph technologies**

* **Neo4j (Community Edition)** — graph database useful for modeling addresses/transactions and running graph algorithms.
* **JanusGraph** (with Cassandra/Elasticsearch backend) — scalable graph database alternative.
* **NetworkX**, **igraph**, **graph-tool** — Python graph analysis libraries for prototyping.

**Wallet / key / recovery & endpoint tools**

* **btcrecover** — password recovery tool for Bitcoin wallets (useful for forensic recovery).
* **pywallet / bitcoinlib / electrum** — libraries and tools that can parse wallet files or recover addresses.
* **Volatility** — memory forensics framework to analyze RAM images (recover keys, wallet artifacts).
* **Autopsy / Sleuth Kit** — disk forensics for seized machines.

**Visualization & analyst tooling**

* **Gephi** — interactive network visualization (desktop).
* **Kibana / Grafana** — dashboards for time-series and logs.
* **Maltego (community)** — OSINT graphing (limited free edition) to pivot on identities/links.

**OSINT and security tools**

* **MISP (Malware Information Sharing Platform)** — store/share IOCs, useful for case management and indicators.
* **Wireshark** — network capture/analysis.

*(Note: many commercial solutions exist—Chainalysis, Elliptic, TRM, Crystal; OSS stack here gives a base you can control and build on.)*

**3) Documentation & How to Start Building a Custom Crypto-Forensics Solution**

**A. Goals & Use Cases (pick 2–4 to start)**

* Transaction tracing & fund flow visualization (follow a stolen fund)
* Exchange address detection & linking (identify deposits to common exchanges)
* Alerting for sanctioned addresses / money laundering indicators
* Wallet/key recovery (endpoint/memory analysis)
* Case management & reporting for investigators

**B. High-level Architecture (proposed)**

[Blockchain Nodes (Bitcoin Core, Geth/Erigon)] <-- authoritative data

|

v

[ETL Layer] (blockchain-etl / custom parsers) -> store raw events -> Parquet/DB

|

v

[Normalization / Enrichment]

- address clustering (BlockSci/GraphSense)

- tag enrichment (OFAC, known wallet labels, exchange IPs)

- reverse-lookups (DNS, WHOIS, OSINT)

|

v

[Storage]

- Graph DB (Neo4j or JanusGraph)

- Time-series / relational (Postgres, Timescale) for balances, flows

- Search index (Elasticsearch)

|

v

[Analytics / ML]

- heuristics, rule engine (suspicious flow, mixing detection)

- clustering & anomaly detection (scikit-learn / PyTorch)

|

v

[API & UI]

- REST API (FastAPI) + Web UI (React + D3) for visualization & case management

|

v

[Alerts & Reporting]

- email/slack + PDF forensic reports, evidence logs, case management (MISP/Custom)

**C. Recommended Languages & Frameworks**

* **Python** — primary choice for ETL, analyst scripts, ML, and orchestration. Rich blockchain libraries and scientific stack (pandas, scikit-learn, networkx).
* **Rust or Go** — for high-performance parsers or stream processors (low latency ingestion).
* **Neo4j** (Cypher) or **JanusGraph** for graph storage/queries.
* **FastAPI** (Python) for REST services (fast, async).
* **React + D3.js / Cytoscape.js** for interactive graph visualizations.

**D. Data Sources & Enrichment**

* Run **full nodes** (Bitcoin Core, Geth/Erigon) for canonical chain data.
* Use **blockchain-etl** to export to Parquet/CSV/BigQuery for analytics.
* Integrate **exchange deposit addresses** via public APIs (or commercial feeds) and sanction lists (e.g., OFAC) for tagging.
* OSINT feeds (darknet marketplaces, takedown lists) to add context.

**E. Core Features to Implement (MVP)**

1. **Ingestion module**: stream new blocks/transactions into processing queue (Kafka recommended).
2. **Normalization**: canonical representation of addresses, scripts, token transfers.
3. **Clustering heuristics**: implement common heuristics (input clustering, change address detection).
4. **Graph model**: load entities/edges into Neo4j; implement queries for path tracing and degree centrality.
5. **Basic UI**: trace funds UI with node/edge visualization, address tagging, timeline playback.
6. **Rule engine & alerts**: detect flows to sanctioned addresses, mixing services, rapid high-value transfers.

**F. Analytics & ML Ideas**

* **Unsupervised clustering** (k-means, DBSCAN) to find anomalous addresses / clusters.
* **Graph-based anomaly detection**: detect nodes with unusual in/out patterns.
* **Classifier**: supervised model to score addresses for risk (requires labeled data).

**G. Development Steps & Timeline (example, 12 weeks MVP)**

1. **Week 0–1**: Requirements, legal checks, environment setup (nodes).
2. **Week 2–3**: ETL + basic ingestion pipeline; store raw blocks.
3. **Week 4–5**: Implement normalization & address index; import into DB.
4. **Week 6–7**: Clustering heuristics; basic graph queries (trace path).
5. **Week 8–9**: Frontend wireframe + API; visual trace tool (React + D3).
6. **Week 10**: Alerting and report generation.
7. **Week 11–12**: Testing, documentation, demo & iteration.

**H. Tools & Stack Summary (one-liner)**

* **Ingestion**: Bitcoin Core / Geth + blockchain-etl (Python)
* **Processing**: Python (pandas, networkx), Kafka (stream), Docker/K8s deployment
* **Storage**: Neo4j (graph), Postgres/Timescale (metadata), Elasticsearch (search)
* **API & UI**: FastAPI + React + D3/Cytoscape.js
* **ML**: scikit-learn / PyTorch
* **Forensics**: Volatility (memory), Autopsy (disk), btcrecover (wallet recovery)

**I. Evidence Handling, Legal & Ethical Considerations**

* Implement **audit trails** & immutable logs for all analyst operations (who viewed/exported data).
* Store chain-of-custody metadata when handling seized devices and recovered keys.
* Ensure **privacy & compliance** (GDPR / local laws): only collect/retain required PII, anonymize when possible.
* Define SLAs and governance for use (who can run traces, disclosure policies).