

Sto facendo un covert channel icmp in cui nascondo i dati nel payload. L'idea era di non mettere i dati raw nel payload ma nascondendoli un po. L'idea era associare ad ogni possibile byte una parola. Per esempio CANE=01011010. Illustrami e citami tutte le fonti che trovi su questo argomento.

Ragionato per 15s >

Ottimo — argomento interessante e pratico per la tesi. Qui sotto ti do:

1. Una spiegazione chiara e completa del metodo «mappare ogni byte (o blocco di bit) su una parola» (pro, contro, varianti pratiche, come costruire la wordlist, capacità e rischi di rilevamento).
2. Una lista ragionata di fonti (paper, articoli e risorse tecniche) che ho trovato — con brevi note su cosa contiene ciascuna fonte e perché è rilevante per il tuo approccio — così puoi leggere e citare direttamente.

1) Schema del metodo e varianti pratiche

Idea base

- Definisci una *wordlist* con 256 voci distinte (o una dimensione diversa se codifichi gruppi di bit). Ogni voce corrisponde a un valore byte (es. CANE → 0x5A = 01011010). In trasmissione mandi sequenze di parole (testo) dentro il payload ICMP invece dei byte raw. Questo consente: (a) modificare la forma del dato inviato per sembrare testo; (b) applicare grammatica / dizionario per rendere il testo più naturale.
- Decodifica: il ricevitore splitta le parole e le mappa indietro ai byte con la stessa tabella.

Varianti e miglioramenti pratici

- **Mapping diretto byte→parola (tabella fissa 256):** semplice, deterministico, basso overhead di implementazione ma produce testo che spesso sembra casuale (se le parole sono scelte arbitrariamente).
- **Mapping a blocchi (es. 12 bit → 4096 parole):** aumenta efficienza rispetto a token piccoli ma richiede cataloghi più grandi.

- **Compressione + mapping:** prima comprimi il file (es. zlib/LZMA) poi converti il flusso bit in parole: aumenta capacità utile e riduce lunghezza della trasmissione, ma la compressione cambia la distribuzione statistica dei simboli ed è più facilmente rilevabile se non mascherata bene.
- **Codifica entropica / Huffman / Shannon-Fano:** puoi associare parole più corte o più plausibili a sequenze più probabili (se hai distribuzione nota), aumentando efficienza ma introducendo complessità e potenziali pattern.
- **Generazione testuale plausibile (linguistic steganography):** usa modelli Markov o LM (o un language model) per generare frasi che incorporano le parole-mappa in contesto naturale; riduce la rilevabilità testuale, ma aggiunge complessità e richiede allineamento strettissimo tra encoder/decoder. Fonti classiche su questo approccio lo descrivono e valutano (vedi sez. fonti). 41 JALIO +1

Pro e contro (riassunto)

- Pro: facilita l'invio di payload che appaiono «testuali», può combinarsi con tecniche di offuscamento linguistico e permette di evitare invio di dati binari riconoscibili.
- Contro: testo "random" o uso anomalo di parole può essere individuato da analisi statistica / modelli NLP; throughput è generalmente basso (sovraccarico semantico); sincronia e gestione di perdita/ritrasmissioni richiedono protocollo applicativo (sequencing, checksum, framing). Rilevabilità e contromisure sono ben documentate per canali ICMP. RIT Digital Repo... +1

Aspetti pratici di implementazione (ICMP)

- Inserisci il testo (sequenza di parole separate da uno spazio o carattere speciale) nel payload di Echo Request/Reply. Calcola MTU e frammentazione: un payload troppo grande viene frammentato e può essere intercettato o alterato.
- Aggiungi intestazioni applicative minime (magic bytes, lunghezza, CRC) prima della codifica in parole per permettere sincronizzazione e rilevamento errori.
- Se vuoi che il traffico sembri "normale", mimetizza: usa parole appartenenti a un lessico coerente (es. frasi naturali), regola il rate di invio e varia lunghezza pacchetti/pause per imitare pattern legittimi (es. ping diagnostici). Tuttavia, ciò non elimina la possibilità di rilevamento tramite modelli ML/statistici. Medium +1

Rilevamento e mitigazioni (cosa guardano gli IDS/ricercatori)

- Analisi statistica della distribuzione delle parole vs. distribuzioni attese (lingua naturale) — se il testo appare troppo casuale, raise a flag.
- Modelli supervised (SVM/Random Forest/NN) addestrati su payload ICMP "normali" vs. covert mostrano buone performance. Molti lavori usano caratteristiche come entropia del payload, n-gram delle parole, lunghezza media e varianza. ResearchGate +1

- Ricorda: costruire e testare covert channels su reti che non sono tue o senza autorizzazione è illegale/contro le policy di molti atenei e aziende. Limita i test ai tuoi dispositivi e alle reti di laboratorio o ottieni permessi espliciti. (Nota importante per la tesi: documenta le autorizzazioni e il contesto degli esperimenti.)

2) Fonti trovate (con annotazioni — leggi queste prime se vuoi capire l'approccio teorico e pratico)

Nota: sotto trovi una selezione ragionata delle fonti più rilevanti che descrivono metodi di text/word mapping, covert channels su ICMP, rilevamento e tecniche di "linguistic steganography". Ho incluso paper accademici, preprint/arXiv e risorse tecniche pratiche.

1. *Image Steganography by Mapping Pixels to Letters* — Al-Husainy (2009).
→ Propone mapping bit→lettere/parole; utile per comprendere l'idea generale di mappare simboli di copertura a bit e i problemi di ridondanza e capacità.
Semantic Scholar
2. *Text Steganographic Approaches: A Comparison* (ArXiv / survey).
→ Confronta approcci di steganografia testuale: generazione, sostituzione di parole, grammatiche, modelli Markov; ottimo per capire pro/contro del mapping parola→bit. arXiv
3. *An Approach for Text Steganography Based on Markov Models* — paper che spiega come usare catene di Markov per generare testo mascherato. Descrive chiaramente anche l'idea di «mappare un blocco di bit su una parola» e le implicazioni sintattiche. 41 JAIIO
4. *Covert Channels with Sequential Transducers* (Benattar et al.).
→ Tratta formalmente encoder/decoder per canali covert e mostra esempi di codifica di bit in simboli di alto livello (utile per la base teorica del mapping). lacl.fr
5. *ICMP Covert Channel Resiliency / repository RIT* (tesi/technical report).
→ Analisi pratica su come si costruiscono e difendono covert channel su ICMP; utile per scenari reali e per capire i limiti/detection. RIT Digital Repo...
6. *Covert Channel Detection in the ICMP Payload Using Support Vector Machine* (ResearchGate) / lavori su SVM per ICMP.
→ Descrive feature e pipeline per distinguere payload ICMP "benigni" da quelli che trasportano dati nascosti. Importantissimo per valutare probabilità di essere scoperti. ResearchGate +1

7. *From Ping to Malware: How ICMP Becomes a Silent Backdoor* — articolo tecnico / blog (Medium).
→ Buona panoramica pratica e divulgativa su come ICMP è stato utilizzato per tunnel/covert channels; utile se vuoi esempi concreti e tool. Medium
8. *Covert channel by exploiting error-correcting codes* (ScienceDirect, 2025).
→ Metodo recente che sfrutta ECC per costruire covert channels; interessante confronto metodologico (diverso meccanismo, ma utile per capire nuove idee su codifica/robustezza). ScienceDirect
9. *A novel steganography technique by mapping words with LSB array* (ResearchGate).
→ Combina concetti di mapping parola→bit con tecniche LSB; utile per vedere ibridi tra steganografia binaria e testuale. ResearchGate
10. Risorse pratiche e tutorial su steganografia (LSB, implementazioni, blog e repository) — es. GeeksforGeeks, Medium, tutorial vari su GitHub/BetterProgramming.
→ Non accademico, ma utile per esempi di codice e implementazioni che puoi adattare per test sul payload ICMP. GeeksforGeeks +1
-

Come citarle nella tesi

- Per i lavori accademici (arXiv / paper / riviste) usa la citazione standard BibTeX; per blog/tutorial cita come "sorgente web" e indica data di accesso.
 - Se vuoi, posso prepararti direttamente le entry BibTeX per i paper principali (3–8 fonti accademiche) in stile `plainnat` o `IEEE` — dimmi lo stile e te le do subito.
-

Se vuoi che proceda con del codice (opzionale, pronto subito)

Posso generare direttamente (qui e ora, senza bisogno di aspettare):

- uno script Python che:
 - costruisce una *wordlist* di esempio (256 parole),
 - codifica un file binario in sequenze di parole e mette il testo risultante nel payload di pacchetti ICMP (usando `scapy`) — *solo per test su host locali o laboratorio*,
 - decodifica dal payload di ICMP ricevuti indietro.
- oppure uno script che genera frasi "più naturali" usando un Markov chain builder che inserisce le parole-mappa in contesto.

Dimmi quale preferisci e te lo produco subito (ti do anche commenti su come usare in laboratorio in modo eticamente corretto).

Se vuoi, posso anche:

- preparare le entry BibTeX per le fonti accademiche sopra elencate (dimmi lo stile), oppure
- generare subito lo script Python di esempio per l'encoding/decoding (incluso un piccolo wordlist demo).

Fammi sapere cosa preferisci — intanto le fonti sopra sono quelle più direttamente rilevanti e complete che ho trovato.

     ...  Fonti

vorrei anche che le parole anche se casuali avessero un senso. Illustrami il concetto che voglio sviluppare e anche qui illustrami tutto ciò che riesci trovare siti, internet, paper ecc

Ragionato per 13s >

Perfetto — l'idea è molto interessante e ci sono varie tecniche per far sì che le parole **anche se casuali abbiano un senso** (cioè risultino plausibili/naturali) pur trasportando bit. Di seguito ti do:

1. una spiegazione dettagliata e pratica del concetto che vuoi sviluppare (varianti, pro/contro, come costruire la wordlist e incastonarla in frasi naturali);
2. indicazioni pratiche per implementare e valutare il sistema (sincronizzazione, framing, throughput, metriche di detection);
3. una raccolta ragionata di paper, articoli e risorse utili — con note su cosa contiene ciascuna fonte (ti ho cercato materiali accademici e tecnici recenti su Markov/Linguistic Steganography, generative stego con modelli di linguaggio, e detection per ICMP).

Procedo direttamente e concreto — niente domande di chiarimento: faccio un best-effort completo sulla base della tua idea.

Concetto centrale che vuoi sviluppare

Vuoi mappare sequenze di bit (es. byte) su parole, ma fare in modo che la sequenza di parole **appare**» come testo con senso (coerente, grammaticalmente accettabile o almeno plausibile) — invece di una lista di parole scollegate. Questo richiede due componenti principali:

1. **Codifica**: una tabella (o algoritmo) che associa blocchi di bit a parole o token (es. $0x5A \rightarrow \text{CANE}$).
2. **Cover text generation / smoothing**: un meccanismo che riordina/arrangia/inserisce quelle parole in un contesto linguistico che sembri naturale (frasi, pseudo-frasi) oppure che genera frasi che incorporano le parole mappa in modo coerente.

Due grandi famiglie di approcci per ottenere il “senso”:

A. **Modifica-based + mapping fisso** — generi frasi usando una grammatica / template e inserisci parole mappa in posizioni predefinite (es. sostituisci aggettivi o sostantivi in template di frasi). È semplice, controllabile e veloce, ma il throughput è basso (pochi bit per parola) e serve un buon dizionario tematico. (v. lavori su mapping parole→bit e sui metodi semplici). [41 JAIIO +1](#)

B. **Generative (Markov / Language Model) con embedding via ranking/binning** — usi un modello statistico (Markov chain, n-gram, o un LLM moderno) per generare testo plausibile; l'algoritmo di embedding sceglie fra un insieme di candidate parole per ogni step in funzione dei bit da nascondere (es. Huffman o “rank-token mapping” dove l'ordine di preferenza dei token codifica informazioni). Questo dà testi molto più naturali e throughput migliore (specialmente con LLM), ma richiede sincronizzazione e l'accesso allo stesso modello/statistiche in ricezione. (v. Markov+Huffman e recenti lavori su LLM-driven stego). [arXiv +1](#)

Modi pratici per far “avere un senso” alle parole-mappa

Ecco tecniche concrete, ordinate da più semplici a più sofisticate:

1. Wordlist tematica + template frasale

- Costruisci una wordlist ragionata per dominio (es. vocabolario culinario, tecnico, sportivo). Per ogni byte scegli una parola dal tema.
- Per trasmettere, inserisci le parole in template (“Ieri al __ abbiamo provato __.”). In ricezione estrai le parole dalle posizioni note.
- Pro: implementazione semplice; Con: testo ripetitivo e throughput basso.

2. Bins / Bucket mapping + grammatica

- Raggruppi parole semanticamente affini in “bucket” (es. 16 bucket × 16 parole → 8 bit per parola con ridondanza).

- Per ogni campo di bit scegli una parola dal bucket corrispondente e poi applichi una grammatica (ad es. accordo sing/plurale, scelta di articolo) per mantenere correttezza.
- Migliora naturalezza e offre possibilità di variare forma lessicale.

3. Markov chain + Huffman / codice adattivo

- Allena una Markov chain su un corpus del dominio; ad ogni stato ottieni una lista di candidate con probabilità condizionate. Applica Huffman sul set di candidate per mappare bit a parole: parole più probabili hanno codici più corti. Questo mantiene la fluidità del testo perché scegli parole che il modello già considera probabili. Implementazioni e reference esistono (Markov+Huffman).

arXiv +1

4. Rank-token / entropy-driven mapping con LLM (modernissimo)

- Usare LLM: per ogni posizione chiedi al modello i token candidati ordinati per probabilità; usi la posizione/rank del token per codificare bit — ad esempio, se i primi 8 token rappresentano 3 bit ciascuno, scegli il token il cui rank corrisponde ai bit da inviare. Tecniche più avanzate usano normalizzazione dell'entropia e compressione per mantenere naturalezza elevata (es. "rank-token mapping", RTMStega-like). Questo è lo stato dell'arte per linguistic stego con alta naturalità e throughput, ma è più complesso da implementare (e richiede che il decoder abbia accesso a un modello con comportamento prevedibile).

arXiv +1

5. Paraphrase / pivot translation + semantic bins

- Applichi traduzioni intermedie o parafrasi per ottenere testi diversi che includono le parole mappa. Alcuni lavori usano "pivot translation" o bins coding per mantenere il significato mentre variano le parole per codificare bit. Utile per mantenere semantica e bypassare alcuni rilevatori lessicali.

arXiv

Implementazione pratica (per ICMP payload)

- **Framing:** prima della conversione in parole, aggiungi header applicativo (magic bytes, versione, lunghezza, seq number, CRC). Codifica l'header stesso nello stesso schema o in un prefisso binario "semplice" per sincronizzare.
- **Compressione:** comprimi prima (zlib/LZMA). Riduce bit da nascondere ma cambia la distribuzione; con schemi generativi può migliorare throughput.

ScienceDirect

- **Rate shaping:** invia pacchetti a ritmi simili al traffico legittimo (es. ping occasionali). Varia tempi e dimensione per evitare pattern periodici.
- **MTU e frammentazione:** mantieni payload entro MTU per evitare frammentazione IP (la frammentazione modifica/espone pacchetti).

- **Decoder:** ricevitore deve usare la stessa wordlist/algoritmo e lo stesso modello (o una versione compatibile). Per LLM-based, devi assicurare che il decoding usi lo stesso checkpoint o che la tecnica sia robusta alle differenze di modello.
- **Gestione errori:** seq numbers + ACK o forward-error-correction (FEC) a livello applicativo per tollerare pacchetti persi.

Valutazione e detection — metriche e contromisure

- **Metriche di naturalezza:** perplexity, BLEU/ROUGE vs corpus tematico, similarity semantica, test umano. Per LLM-generated stego, misura perplexity rispetto a un modello di lingua di riferimento. [arXiv](#)
- **Detection features (usati da IDS/ricercatori):** entropia del payload, distribuzioni di n-gram, lunghezza media/varianza parole, rate e periodicità, incongruenze semantiche. Lavori mostrano che SVM/RandomForest/NN su queste feature identificano covert ICMP. [ResearchGate](#) +1
- **Test di robustezza:** cross-validate il decoder usando versioni diverse del corpus/modello e aggiungi rumore (drop di pacchetti, ritardi).
- **Trade-off sicurezza vs throughput:** più naturale = spesso minore throughput (o più complesso embedding). LLMs tendono a offrire miglior trade-off ma richiedono infrastruttura.

Aspetti etici e legali (da non sottovalutare)

Test su reti altrui o produzione senza autorizzazione è rischioso/illegale. Per la tesi documenta ambiente di test, permessi e misure di contenimento. Questo è fondamentale per pubblicare/valutare risultati senza problemi etici o legali.

Fonti principali trovate (con breve nota)

Di seguito le risorse che ho identificato come più rilevanti per il tuo specifico approccio (mapping parola↔bit e generative linguistic steganography), ordinate per utilità pratica e teorica:

1. Yang, Z. et al., “Automatically Generate Steganographic Text Based on Markov chain model and Huffman coding” — descrive chiaramente l'accoppiamento Markov+Huffman per generare testo stego (codifica token→bit). Ottimo come punto di partenza per implementazioni Markov. [arXiv](#)
2. *An Approach for Text Steganography Based on Markov Chains* (Moraldo / [arXiv](#) / implementazione di riferimento) — presenta metodi Markov per generare testi con bit incapsulati; include riferimento a implementazioni. Utile anche come base per esperimenti. [arXiv](#)

3. **"Semantic-Preserving Linguistic Steganography by Pivot Translation and Semantic-Aware Bins Coding"** — paper che affronta come mantenere la semantica mentre si codificano bit (pivot translation, bins coding). Utile per la parte "far avere senso" alle parole. [arXiv](#)
4. **RTMStega / Entropy-driven rank-token mapping** (lavori recentissimi su LLM-driven stego; articoli 2024–2025) — tecniche che usano ranking dei token e normalizzazione dell'entropia per una steganografia ad alta capacità con text naturalness elevata. Indispensabile se vuoi usare LLM per generare testo plausibile che nasconde molti bit. [arXiv](#) +1
5. **Survey / confronto degli approcci testuali** — "Text Steganographic Approaches: A Comparison" (Agarwal et al., survey) fornisce panoramica metodologica e valuta pro/contro delle tecniche testuali tradizionali. Buono per la parte "stato dell'arte" nella tesi. [arXiv](#)
6. **ICMP-specific: "ICMP Covert Channel Resiliency" (RIT repository) e altri report tecnici** — esamina covert channel su ICMP e criticità pratiche (framing, detection, mitigazioni). Da leggere per la parte sperimentale sulla rete. [RIT Digital Repo...](#) +1
7. **Detection papers** — "Covert Channel Detection in the ICMP Payload Using Support Vector Machine" e lavori su attacchi e detection testuale (feature-based). Importanti per valutare la probabilità che il tuo canale sia scoperto. [ResearchGate](#) +1
8. **Introduzioni / survey su Linguistic Steganography** — "Introduction to Linguistic Steganography" e altre review che coprono modificazione-based e generative-based. Utile per capitolo introduttivo della tesi. [ResearchGate](#)
9. **Articoli pratici e blog tecnici** — spiegazioni divulgative su ICMP tunnels e esempi (utile per implementazioni di prova e tool esistenti). Non accademico ma pratico. [Wikipedia](#) +1

Suggerimenti pratici e piano in 5 passi (per prototipo della tesi)

1. **Scegli dominio** (es. notizie sportive, cucina) e costruisci corpus (1–10 MB) — serve per allenare Markov/estrarre candidate token.
2. **Implementa versione basica**: mapping byte→parola con template (proof-of-concept), incapsula testo in payload ICMP e verifica decodifica in locale.
3. **Markov+Huffman**: implementa la versione con Markov + Huffman coding per aumentare naturalezza e throughput. Testa perplexity e detection semplice. [arXiv](#)

4. **LLM-driven** (se hai risorse): prova rank-token mapping su un piccolo modello locale (o API) e valuta naturalità vs throughput; misura come cambia la detectability. arXiv
 5. **Valutazione**: monta dataset di ICMP benigni e stego, misura AUC di detection, perplexity, accuratezza di decodifica in presenza di perdite.
-

Se vuoi, ti posso **fornire subito** (qui e ora, senza aspettare):

- uno **script Python** demo che implementa la pipeline *Markov + Huffman* (generazione di cover testuale che incorpora bit) e un decoder (no invio raw ICMP, ma puoi integrarlo con `scapy` per test in laboratorio); oppure
- un **progetto di esempio** che implementa la versione a template e la versione Markov, con i comandi per allenare la Markov chain sul corpus che scegli; oppure
- preparare per te le **entry BibTeX** per i paper accademici che ho citato (stilizzazione in `plainnat` o `IEEE`) così le incorpori subito nella tesi.

Dimmi quale preferisci e te lo consegno subito (codice e/o BibTeX).